UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

DEPARTAMENTO DE INGENIERÍA TELEMÁTICA

PROYECTO FIN DE CARRERA

Búsqueda geoposicional de redes WiFi con dispositivos Android

Autor:
Dña. Patricia Bravo García.

Departamento:
Ingeniera Telemática.

Tutor:
D. Vicente Luque Centeno.

Tribunal de Evaluación:

Presidente: Dña. Florina Almenárez Mendoza.

Secretario: D. Manuel Urueña Pascual.

Vocal: D. Luis Emilio García Castillo.

Fecha de lectura: 04 de Febrero de 2010

Calificación:

UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA DE TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA

Búsqueda geoposicional de redes WiFi con dispositivos Android

PATRICIA BRAVO GARCÍA

MADRID, 2010

Agradecimientos

Quiero agradecer y dedicar este trabajo a mi familia, especialmente a mis padres, mi hermano y mis abuelos. Gracias por creer en mí y estar a mi lado siempre.

Mención destacada merecen también mis amigos y compañeros de clase, así como todos los profesores que en algún momento han contribuido a mi formación. En este punto quiero expresar mi agradecimiento a Vicente Luque Centeno por el trato recibido durante el desarrollo de este Proyecto Fin de Carrera.

Y por supuesto, a Isaac. Gracias por tu apoyo incondicional, este trabajo es también triunfo tuyo.

A todos aquellos que me habéis ayudado a llegar hasta aquí, MUCHAS GRACIAS.

Resumen

El presente Proyecto Fin de Carrera, perteneciente a la titulación de Ingeniería Superior de Telecomunicaciones de la Universidad Carlos III de Madrid, tiene como finalidad implementar un sistema de búsqueda de redes WiFi. Este sistema consta de una aplicación web 2.0 y una aplicación para dispositivos móviles Android. Desde dichos terminales será posible encontrar puntos de acceso WiFi para que el usuario se pueda conectar si así lo desea.

Para implementar el primero de los módulos se ha utilizado la tecnología Flex. Dicha tecnología permite realizar aplicaciones ágiles e intuitivas para el usuario a costa de los recursos del cliente, consiguiendo descargar la transferencia de datos por parte del servidor.

La aplicación web dispone de un mapa virtual para que los usuarios puedan gestionar los puntos de acceso (APs) WiFi disponibles. Según su rol los usuarios podrán añadir, modificar, eliminar APs, importar APs de otros sistemas, etc. La aplicación también permite buscar el punto de acceso más próximo a una región específica, mostrando todos sus detalles (latitud, longitud, potencia de transmisión, velocidad, etc).

Para insertar mapas interactivos en la herramienta ha sido necesario utilizar el API de Google Maps, que proporciona varias utilidades para manipular y añadir contenido a dichos mapas.

Por otro lado, en este PFC se ha estudiado el funcionamiento y las posibilidades que ofrece Android, comenzando por un enfoque analítico de sus características hasta llegar al desarrollo de la aplicación que permite buscar puntos de acceso WiFi desde este tipo de dispositivos. La aplicación desarrollada puede funcionar de dos modos distintos:

- Cada vez que un usuario entra en el área de cobertura de un AP, en la pantalla del terminal Android se muestra información de dicho punto de acceso para que el usuario se pueda conectar a él si así lo desea.
- Un usuario puede buscar un punto de acceso WiFi aunque no se encuentre dentro del área de cobertura en base a los siguientes criterios:
  - Por proximidad a la localización actual del usuario.
  - Por proximidad a una región específica.
  - Por máxima potencia (en una distancia máxima).
  - Por máxima velocidad (en una distancia máxima).

La solución propuesta en este PFC se puede enmarcar en el ámbito del software colaborativo, pues pretende que todos los puntos de acceso WiFi (independientemente del usuario por el que hayan sido añadidos) sean accesibles por todos los usuarios.

Abstract

This project aims to implement a search of WiFi networks. The system is composed of a web 2.0 application and an application for Android mobile devices. From these terminals will be possible to find WiFi access points (APs) so that the user can be connected if desired.

To implement the first module Flex technology has been used. With this technology is possible developing flexible and intuitive applications for the user.

The web application has a virtual map so that users can manage the APs WiFi available. According to his role users can add, modify, delete APs, import APs from other systems, etc. Also, the application allows to find the nearest AP to a specific region, showing all the details (latitude, longitude, transmission power, speed, etc).

To insert interactive maps has been necessary to use the Google Maps API, which provides several utilities to manipulate and add content to these maps.

Moreover, in this PFC Android technology has been studied. An application that helps the user find WiFi APs from these devices has been developed. Two modes exist:

- Each time a user enters within an AP coverage area, some information of this AP is shown. The user can connect to it if desired.
- A user can find a WiFi AP (even if he is not within the coverage area) based on the following criteria:
  - For proximity to the user's current location.
  - By Proximity to a specific region.
  - For maximum power (at a distance).
  - For maximum speed (at a distance).

The proposed solution can be framed in the field of collaborative software, all WiFi access points (regardless of the user who has added them) are accessible for all users.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

A lo largo de esta memoria se pueden descubrir todos los detalles y las bases teóricas de la implementación del Proyecto Fin de Carrera "Búsqueda geoposicional de redes WiFi con dispositivos Android". Se ha implementado un servicio de descubrimiento de puntos de acceso Wi-Fi (en adelante **WiFi Locator**) compuesto de una aplicación web (basada en tecnología Flex y el API de Google Maps) y una aplicación para dispositivos móviles Android.

El presente capítulo de introducción consta de seis apartados.

- El primero expone los motivos que justifican la realización de este PFC.
- El segundo recoge los objetivos a conseguir.
- En el tercer bloque se presenta en detalle la estructura de la memoria.
- El cuarto segundo presenta un glosario de los términos utilizados.
- En el quinto apartado se realizará un estudio de la tecnología WiFi.
- Dado que en este PFC se han empleado tecnologías de libre distribución, en el último apartado se hará una reflexión sobre el concepto de software libre.

1.1. Motivación

Ahora que la mayoría de los dispositivos electrónicos del mercado tienen conexión WiFi, y que cada vez son más los sitios públicos (bares, centros comerciales, etc) que disponen de este tipo de redes, se hacen muy necesarios los buscadores de puntos de acceso (AP) WiFi. Para registrar dichos APs es de gran utilidad disponer de una herramienta basada en Google Maps. Así, un usuario puede ubicar los puntos de acceso de forma precisa y conocer detalles como la potencia de tranmisión, velocidad, etc.

Tal y como se ha mencionado anteriormente, la solución propuesta en este PFC para la búsqueda de redes WiFi (**WiFi Locator**) se compone de una aplicación web y una aplicación móvil.

Sobre la parte web es de resaltar que en los últimos años se ha producido un crecimiento muy grande en cuanto a usabilidad; disfrutamos cada día de aplicaciones web que se aproximan mucho a aplicaciones de escritorio, evolucionando de forma constante para satisfacer las necesidades de los usuarios (interfaces más intuitivas, etc) sin esperar a la finalización de ciclos cerrados de desarrollo (como ocurre por ejemplo en los principales sistemas operativos de carácter privativo). Todo este auge ha dado sentido a una nueva visión de la red, no sólo existe la necesidad de procesar la información que existe sino que el usuario está interesado en crearla.

Por otro lado, la tecnología móvil ha avanzado de una manera espectacular esta última década. Con los primeros móviles de los 90 parecía imposible pensar que algún día la gente pudiera conectarse a Internet, escuchar música, etc. Pues bien, hoy en día ya es algo común y al alcance de todo el mundo. Pero no está todo inventado, al menos las empresas se esfuerzan por hacer ver que no es así. Apple con su IPhone ha sorprendido al público con un teléfono con pantalla multitáctil, que reproduce audio, vídeo, etc. Nokia está pensando en móviles basados en la nanotecnología. Symbian que está preparando su primera versión "libre", la s60 y ahora Google, que ha decidido conquistar también el mundo de la tecnología móvil con Android, el primer sistema operativo libre, de código abierto y que provee servicios basados en la localización geográfica.

Para la implementación de este PFC se han empleado varios conceptos y tecnologías disponibles en el momento actual. Se han escogido aquellas con las que el desarrollo es más correcto, estructurado y eficiente.

1.2. Objetivos

1.2.1. Objetivos formales

- Creación un interfaz web para la gestión de puntos de acceso WiFi integrada con las características de Google Maps
  La interfaz web se integrará con el servicio Google Maps a través del API correspondiente. Google Maps es un servicio de Google que permite visualizar de mapas de todo el mundo, seleccionar una calle o un punto en el mapa, etc.

- Diseño de una interfaz para dispositivos Android para la búsqueda de puntos de acceso WiFi
  Se podrán realizar búsqueda de redes WiFi con terminales Android en base a cuatro criterios: por proximidad a la posición actual, por proximidad a una región específica o

por potencia/velocidad (en una distancia máxima).

1.2.2. Objetivos operacionales

- La aplicación (componente web y componente Android) debe ser sencilla e intuitiva para los usuarios, inclusive para aquellos con escasos conocimientos informáticos. Para ello se diseñará una interfaz gráfica atractiva y fácil de usar y que deberá cumplir los siguientes subobjetivos:
  1. Uniformidad en las páginas, conservando la localización de menús, logotipo, tipo de letra, etc.
  2. Los nombres deben ser intuitivos, facilitando así la comprensión de para qué sirve cada opción del menú.
- El componente web ha de tener un funcionamiento multiplataforma, esto es, debe ser posible explotarlo con independencia de la plataforma utilizada.
- El sistema ha de ser escalable, es decir, el código (del componente web y del componente Android) debe ser comprensible y modular para una posible modificación o ampliación en futuros proyectos.
- El coste del sistema debe ser lo más reducido posible, intentando hacer uso de herramientas de software libre.
- El producto final debe ser confiable y eficiente, haciendo el mejor uso posible de los recursos del sistema operativo en que se ejecute.

1.3. Glosario de términos

- ADSL: Asymmetric Digital Subscriber Line
- AP: Access Point
- API: Application Programming Interface
- DB: DataBase
- DTD: Document Type Declaration
- EJB: Enterprise Java Bean
- GPS: Global Positioning System
- GUI: Graphic User Interface

- HTML: Hypertext Mark-up Language
- HTTP: Hypertext Transfer Protocol
- IDE: Integrated Development Environment
- IP: Internet Protocol
- JAXP: Java Api for XML Processing
- JDBC: Java Database Connectivity
- JSP: JavaServer Pages
- JVM: Java Virtual Machine
- MVC: Modelo Vista Controlador
- MIDP: Mobile Information Device Profile
- PFC: Proyecto Fin de Carrera
- RPC: Remote Procedure Control
- SDK: Software Development Kit
- WWW: World Wide Web
- XML: eXtensible Markup Lenguaje

1.4. Contenido de la memoria

La estructura del presente trabajo se expone a continuación.

- En el primer capítulo se presentan los objetivos del PFC, el contenido de la memoria y un glosario de los términos empleados. Por otro lado, y ya que todas las tecnologías empleadas son *Open Source*, se exponen las bases del software libre. También se describe la tecnología WiFi.
- En el segundo capítulo se revisan las tecnologías web existentes para el desarrollo de aplicaciones RIA, analizando en detalle la plataforma *Flash* que será la tecnología utilizada para el desarrollo de una aplicación web desde la que poder gestionar, de forma centralizada, puntos de acceso WiFi. También se describen las APIs de Google empleadas para integrar las funcionalidades de Google Maps en la aplicación web.

- En el tercer bloque se realiza un estudio detallado de la tecnología móvil Android, mostrando los recursos que ofrece así como el tipo de aplicaciones que se pueden desarrollar y las herramientas que ayudan a su implementación. Dicha tecnología será la que se utilice para implementar la aplicación de búsqueda de redes WiFi desde dispositivos Android.

- En el cuarto capítulo de este trabajo se describen otras herramientas utilizadas: Tomcat como servidor web, Ant como herramienta para compilar la aplicación web y MySQL como base de datos relacional (compartida por aplicación web y la aplicación móvil).

- En el quinto capítulo se presenta la solución propuesta en este PFC para la búsqueda de redes WiFi (aplicación web y aplicación para dispositivos móviles Android) en la que se aúnan todos los conceptos explicados a lo largo del proyecto. Se detallan algunas de las funciones implementadas en los diferentes módulos.

- En último lugar se realiza un análisis de las soluciones ya existentes para la búsqueda de redes WiFi y se presentan algunas conclusiones alcanzadas y futuras líneas de trabajo.

A continuación se describe detalladamente la tecnología WiFi.

1.5. Redes inalámbricas

Una red WLAN (*Wireless Local Area Network*) es una red de área local inalámbrica implementada como una extensión de una red por cable. Los componentes de una WLAN son:

1. Dispositivos portátiles dotados con tarjetas de red inalámbrica y con un consumo de potencia relativamente bajo. Estos dispositivos se conectan a puntos de acceso (AP) vía señales de radio o infrarrojos.

2. Puntos de acceso que interconectan dispositivos de comunicación inalámbrica.
   Un AP puede conectarse a una red de cable y transmitir datos entre los dispositivos conectados a ella y los dispositivos inalámbricos. También puede estar integrado en el módem-router para que distintos dispositivos compartan la conexión a Internet.
   Un AP puede puede funcionar en un rango de 30 a centenares de metros.

Las implementaciones de las WLANs abarcan todas las modalidades posibles: PANs (*Personal Area Networks*), MANs (*Metropolitan Area Network*) y WANs (Wide Area Networks). Las PANs son redes inalámbricas de corto alcance, mientras que las redes inalámbricas tipo WAN y MAN conectan redes de área local utilizando enlaces punto-punto y punto-multipunto.

Los estándares de WLAN son desarrollados por organismos como el IEEE (*Institute of Electrical and Electronics Engineers*) y la ETSI (*European Telecommunications Standards Institute*).

Para resolver este problema los principales vendedores de soluciones inalámbricas crearon en 1999 la asociación WECA (*Wireless Ethernet Compability Alliance*). El objetivo era fomentar más fácilmente la tecnología inalámbrica y asegurar la compatibilidad de equipos. En abril de 2000 WECA certificó la interoperatividad de equipos según la norma IEEE 802.11b bajo la marca WiFi (*Wireless Fidelity*).

A continuación se establece una comparativa de las velocidad de las redes con cables y las redes inalámbricas (WiFi).

- Con cables
  1. Ethernet 10
     La velocidad máxima de transmisión es (aproximadamente) 10 Mbps.
  2. Ethernet 10/100
     La velocidad máxima de transmisión es (aproximadamente) 100 Mbps pero típicamente está entre 20 y 50 Mbps. Es compatible con Ethernet 10.
  3. Ethernet 10/100/1000
     Es la más utilizada porque es diez veces más rápida que la anterior. Se ha empezado a instalar a la par que las redes inalámbricas, por tanto tiene que competir con la versatilidad y facilidad de implantación de éstas. Es compatible con las dos anteriores.
- Inalámbricas
  1. 802.11b
     Esta red ofrece una velocidad máxima de transmisión de 5 Mbps.
  2. 802.11g
     Esta red triplica (aproximadamente) la velocidad de transmisión de la red 802.11b.
  3. 802.11n
     Es el próximo estándar que será compatible con las redes anteriores.

1.5.1. Ventajas de las redes WiFi

Algunas de las ventajas que ofrecen las redes WiFi frente a las redes cableadas son:

- Compatibilidad
  La *WiFi Alliance* asegura que la compatibilidad entre dispositivos WiFi es total.
- Movilidad
  Un usuario puede conectarse a Internet desde cualquier punto del área de cobertura de una red WiFi sin necesidad de hacer llegar ningún cable. La comodidad que ofrecen las redes WiFi es muy superior a la de las redes cableadas porque además el usuario puede desplazarse sin perder la comunicación.

- Flexibilidad
  Extender una red cableada no es tarea fácil; en muchas ocasiones se acaban colocando peligrosos cables por el suelo para evitar poner enchufes de red más cercanos.
  Con las redes inalámbricas (WiFi) no se dan estos problemas, es más, están especialmente indicadas para aquellos lugares en los que se requieren acceso esporádicos a Internet.

- Ahorro de costes
  Diseñar o instalar una red cableada puede llegar a alcanzar un alto coste, no sólo económico, sino también en tiempo. La instalación de una red WiFi permite ahorrar costes.

- Escalabilidad
  Se denomina escalabilidad a la facilidad de expandir la red después de su instalación. Conectar un nuevo dispositivo en una red WiFi es muy sencillo, basta con añadirle una tarjeta. Realizar esta intervención en una red cableada requiere instalar un nuevo cableado. Una vez configuradas, las redes WiFi permiten el acceso de múltiples usuarios sin ningún gasto en infraestructura, no así en la tecnología por cable.

- Portabilidad
  Un usuario puede trasladar su red local sin depender de complejos cableados fijos.

1.5.2. Desventajas de las redes WiFi

Evidentemente, no todo son ventajas, las redes WiFi también tiene puntos negativos en su comparativa con las redes de cable. Los principales inconvenientes son:

- Menor ancho de banda
  Actualmente las redes de cable trabajan a 100 Mbps, mientras que las redes WiFi lo hacen a 11 Mbps. Existen estándares que alcanzan 54 Mbps y soluciones propietarias que llegan a 100 Mbps, pero dichos estándares están en los comienzos de su comercialización y tienen un precio superior al de los actuales equipos WiFi.

- Mayor inversión inicial
  El coste de los equipos de red WiFi es superior al de los equipos de redes cableadas.

- Seguridad
  Las redes inalámbricas no necesitan un medio físico para funcionar. Esto, que aparentemente es una ventaja, es en realidad un inconveniente ya que cualquier persona con un dispositivo portátil puede intentar acceder a la red WiFi (únicamente necesita estar dentro de su área de cobertura).
  Por otro lado, el sistema de seguridad que incorporan las redes WiFi no es demasiado

fiable (aunque ya existe una nueva versión mejorada, WPA) [1], aunque resulta válido para la inmensa mayoría de las aplicaciones.

- Interferencias
  Las redes WiFi utilizan la banda de frecuencias alrededor de 2,4 GHZ del espectro radioeléctrico. Esta misma banda es utilizada por muchos equipos del mercado (teléfonos inalámbricos, microondas, etc) ya que no requiere licencia administrativa. Las interferencias producidas por otros equipos y las pérdidas de señal que el ambiente puede acarrear, afectan negativamente al rendimiento (velocidad) de las redes WiFi.

- Incertidumbre tecnolgóica
  La tecnología que ha adquirido una mayor popularidad es la conocida como WiFi (IEEE 802.11B), pero ya existen otras tecnologías que ofrecen una mayor velocidad de transmisión y unos mayores niveles de seguridad.
  Lo cierto es que las leyes del mercado vienen también marcadas por las necesidades del cliente y, aunque existe una incógnita, los fabricantes no querrán perder el tirón que ha supuesto WiFi y harán todo lo posible para que los nuevos dispositivos sean compatibles con los actuales.

1.5.3. ¿Dónde colocar un AP para proporcionar la mejor cobertura?

En la implantación de un sistema WiFi se debe poner especial atención a factores que afectan a la potencia de la señal recibida, especialmente la ubicación del AP y la interferencia causada por la proximidad de otros equipos WiFi o equipos capaces de emitir señales que produzcan interferencias. Tal y como se comentaba en el apartado anterior, el hecho de operar en una banda libre supone algunas ventajas como el bajo coste y la rápida implantación de la tecnología, pero también presenta inconvenientes como las interferencias.

La respuesta a la pregunta planteada más arriba no es sencilla. La solución viene como resultado de experimentar mediante la técnica de ensayo-error. En cualquier caso, algunos puntos a tener en cuenta son:

1. Colocar el AP tan alto como sea posible para maximizar la transmisión sin problemas de la señal radio.

2. Los sensores (antenas) del punto de acceso no deben ser bloqueados con materiales que puedan afectar a la potencia de la señal emitida.

3. La antena debe estar totalmente perpendicular al suelo para maximizar la transmisión de la señal en todas las direcciones. Colocar la antena inclinada o en posición horizontal reduce la potencia de la señal recibida.

[1] WPA incluye nuevas características de seguridad como la generación dinámica de claves de acceso

4. El punto de acceso debe estar alejado de otros puntos de acceso, especialmente para aquellos casos en los que disponen de canales de transmisión WiFi cercanos ya que éstos son origen de potentes señales de interferencia.

Una elevada potencia de transmisión de señal es importante para proporcionar una buena cobertura (teniendo en cuenta que la la legislación de *Wireless* en España para el nivel de señal en transmisión es de 100mW para la frecuencia de 2.4GHz y de 1W para la frecuencia de 5.4GHz). No obstante, es interesante subrayar que un alto nivel de señal no equivale necesariamente a una mejor conectividad porque las interferencias de otras señales pueden impactar en las prestaciones y estabilidad del sistema WiFi, perdiéndose la conexión.

A continuación se exponen las bases del software libre ya que todas las tecnologías empleadas en este PFC son *Open Source*.

1.6. Software Libre

El movimiento Software Libre comenzó en 1984 cuando Richard Stallman anunció el proyecto GNU. A continuación se presentan dos conceptos que no deben confundirse, *Open Source* y *Free Software*.

1.6.1. Open Source

Según la OSI (*Open Source Initiative*) *Open Source* no significa únicamente acceso al código fuente. Los términos de distribución del Software de Código Abierto deben cumplir los siguientes criterios:

1. Libre redistribución
   La licencia no restringe la venta o cesión del software como componente de una distribución que contenga varios códigos distintos. La licencia no requiere el pago de derechos ni cualquier otra forma de pago por la venta.

2. Código Fuente
   El programa incluye el código fuente y debe permitir su distribución como tal así como en forma compilada. En el caso de que el producto distribuido no incluya el código fuente debe haber una forma suficientemente clara de obtenerlo, por un precio razonable no mayor del coste de reproducción. No se permite el código fuente deliberadamente enrevesado ni tampoco formas intermedias como salidas del preprocesador o traductor.

3. Trabajo derivado

La licencia debe permitir la distribución de trabajos derivados en los mismos términos que el software original.

4. Integridad del código fuente del autor
   La licencia debe hacer explícito el permiso de distribución de software generado a partir de código fuente modificado. La licencia puede requerir que los trabajos derivados lleven un nombre o número de versión diferente al del software original.

5. No discriminación contra personas o grupos de personas
   La licencia no puede discriminar a ninguna persona o grupo de personas.

6. No discriminación contra áreas de trabajo
   La licencia no puede restringir a nadie el uso del software en un campo específico de trabajo.

7. Distribución de la licencia
   Los derechos vinculados al software deben ser de aplicación para todos aquellos a los que haya sido redistribuido, sin necesidad de ejecutar una nueva licencia para aquellas partes.

8. La licencia no debe ser específica para un producto
   Si el software se distribuye de acuerdo a los términos de licencia del programa, todas las partes a las que se distribuye el programa deben tener los mismos derechos que los que están licenciados en la distribución de software original.

9. La licencia no debe restringir otro software
   La licencia no puede plantear restricciones en otro software que es distribuido acompañando al software licenciado.

10. La licencia debe ser tecnológicamente neutral
    Ninguna licencia puede estar dedicada a una tecnología individual.

1.6.2. Free Software

Desde otro punto de vista la FSF (*Free Software Foundation*) define Software Libre de la siguiente manera:

- El Software Libre es un asunto de libertad, no de precio.
- Software Libre se refiere a la libertad de los usuarios para ejecutar, copiar, distribuir, estudiar, cambiar y mejorar el software. De modo más preciso, se refiere a cuatro libertades de los usuarios del software:

  1. La libertad de usar el programa con cualquier propósito.

2. La libertad de estudiar cómo funciona el programa y adaptarlo a sus necesidades. El acceso al código fuente es una condición previa para esto.
3. La libertad de distribuir copias.
4. La libertad de mejorar el programa y hacer públicas las mejoras de modo que toda la comunidad se beneficie. El acceso al código fuente es un requisito previo para esto.

Un programa de Software Libre debe estar disponible para uso, desarrollo y distribución comercial; el Software Comercial Libre es muy importante.

Capítulo 2

Tecnologías web

Desarrollar aplicaciones (de cualquier complejidad) con HTML, CSS y Javascript y obtener resultados idénticos en todos los navegadores es difícil y costoso. Los desarrolladores pueden llegar a pasar el 90 por ciento del tiempo estudiando las peculiaridades de los navegadores.

2.1. Revisión aplicaciones RIA

Las aplicaciones RIA (*Rich Internet Applications*) son aplicaciones web con características similares a las aplicaciones de escritorio. Se apoyan en un desarrollo *cliente-servidor* y no en un desarrollo web tradicional, donde el estado se mantiene en el servidor en sesiones.

- Cliente
  Se maneja la interacción entre el usuario y la *interfaz del usuario*; el usuario invoca comandos, actualiza vistas y carga datos. Aquí se mantiene el estado de la aplicación, se manejan todas las peticiones de datos hacia el servidor y se controla la presentación de los datos.

- Servicios
  Se manejan y se procesan las peticiones del lado del cliente.

Los *applets* de Java se convirtieron en la primera tecnología para el desarrollo de este tipo de aplicaciones. Sin embargo no alcanzaron el éxito debido a, entre otros factores, la necesidad de la instalación de Java en el lado del cliente y a la lentitud de carga de dichas aplicaciones. En este punto surgió Ajax, una tecnología Javascript para la comunicación asíncrona con el servidor, que se hizo popular porque no es necesaria la instalación de ningún software adicional.

En la Figura 2.1 se muestra la arquitectura de una aplicación AJAX comparada con una aplicación web tradicional. Mientras que esta última recarga la página completa con cada petición al servidor, en la aplicación AJAX la página se carga una sola vez, y se actualiza de forma asíncrona con múltiples peticiones ligeras al servidor.

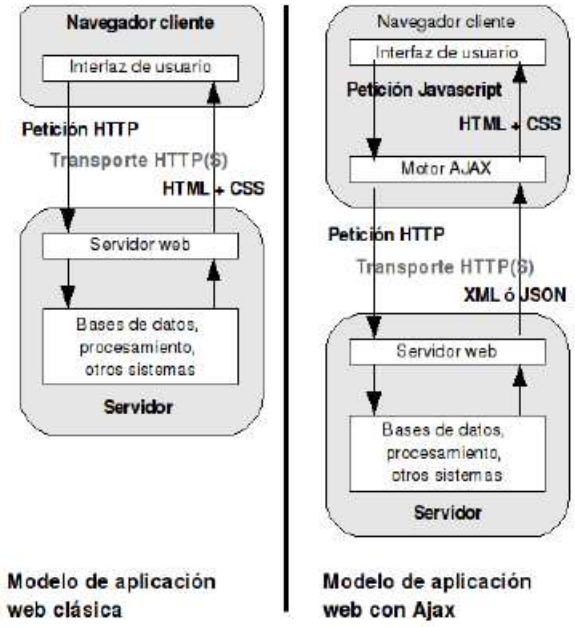

Figura 2.1: Arquitectura aplicación AJAX

En los últimos años han surgido algunas tecnologías que vienen a facilitar el desarrollo de aplicaciones RIA. A continuación se describen las más relevantes.

2.1.1. GWT

Google Web Toolkit (GWT) es un framework creado por Google que permite ocultar la complejidad de varios aspectos de la tecnología web (HTML, Javascript, AJAX). El concepto de GWT es bastante sencillo, la idea es crear código Java empleando cualquier IDE para que el compilador de GWT lo traduzca a código HTML y Javascript optimizado.

Una aplicación GWT puede ser ejecutada en dos modos:

- *Hosted mode*
  Es el modo más usado para desarrollo, soporta el cambio de código en caliente y la depuración.
  Durante el desarrollo de una aplicación se puede repetir el ciclo típico de Javascript (editar, actualizar y ver) y ver inmediatamente los cambios realizados en el código mediante el navegador del modo *Hosted mode*. La ejecución se lleva a cabo en la JVM como bytecode de Java.

- *Web mode*
  Este modo se suele emplear en el despliegue de la aplicación.

En el momento de la implementación GWT compila el código Java en archivos Javascript que se pueden desplegar en cualquier servidor web; por tanto, sólo hay que escribir el código una vez y GWT lo convierte al formato Javascript más adecuado.
A diferencia de los minimizadores de Javascript, que sólo funcionan con texto, el compilador de GWT realiza optimizaciones y un análisis estático completo de toda la base de código GWT y, frecuentemente, genera código Javascript que se carga y ejecuta con mayor rapidez que el código Javascript equivalente creado de forma manual. Por ejemplo, el compilador de GWT suprime de forma segura todo el código que no se utiliza para asegurar que el archivo de secuencias de comandos compilado sea lo más pequeño posible. Otro ejemplo: el compilador de GWT realiza una sustitución en línea de llamadas a funciones en los métodos, lo que permite optimizar el rendimiento de las ejecuciones de métodos.

Componentes

GWT contiene los siguientes componentes:

- *GWT Java-to-Javascript Compiler*: traduce el código desarrollado en Java al lenguaje Javascript. Este componente se utiliza en el modo *Web Mode*.
- *Hosted Web Browser*: ejecuta la aplicación Java en la JVM sin traducirla a Javascript. Este componente se utiliza en el modo *Hosted Mode*.
- *JRE Emulation Library*: contiene las clases más importantes de la librería estándar de Java:
  - **java.lang**: donde se encuentran las clases fundamentales para poder programar en Java, entre otras la clase **java.lang.Object**, que es la clase fundamental de la que heredan o extienden todas las clases en Java.
  - subconjunto de las clases del paquete **java.util**.

  Otras clases no están incluidas en GWT aunque paulatinamente se van incorporando.
- *GWT Web UI Class Library*: contiene un conjunto de elementos de interfaz de usuario que permiten la creación de objetos tales como textos, cajas de texto, imágenes y botones.

Comunicación con el servidor

GWT admite numerosos protocolos de transferencia, como JSON y XML, pero el mecanismo RPC (*Remote Procedure Call*) de GWT permite el establecimiento de comunicaciones Java de una forma especialmente sencilla y eficaz, sólo hay que crear una interfaz que especifique los métodos remotos que se quieran ejecutar. Al realizar una llamada a un método remoto desde

el navegador, el mecanismo RPC de GWT serializa automáticamente los argumentos, ejecuta el método adecuado en el servidor y serializa el valor de retorno.

Ventajas e inconvenientes

Dado que GWT utiliza el lenguaje Java se pueden detectar algunos errores (errores ortográficos, tipos no coincidentes) en tiempo de compilación, no durante la ejecución del programa, lo que aumenta la productividad y reduce los errores. Otra ventaja es que GWT genera automáticamente código compatible con los navegadores más populares (IE, Firefox, Safari, Opera y Google Chrome). De esta forma los desarrolladores no tienen que realizar dicha distinción en su código.

Debido a que el código generado es Javascript, GWT hereda las limitaciones de dicha tecnología: rendimiento, dificultades en la depuración, disponibilidad de librerías, etc.

2.1.2. Plataforma Flash

Después de 10 años Java no ha conseguido dominar el mundo de las RIA. Por otro lado, AJAX sigue presentando las limitaciones impuestas por los navegadores, HTML, Javascript y CSS, que impiden que se expanda más allá de sus límites actuales, aunque también es cierto que se han desarrollado aplicaciones sorprendentes con esta tecnología (como por ejemplo, Google Mail). Sin embargo, requieren un esfuerzo de desarrollo considerable y amplios conocimientos.

Una de las alternativas más recomendables para la programación gráfica de usuario (*Graphical User Interface*) es *Flash*. Las aplicaciones basadas en esta tecnología dependen de un *plug-in* del navegador; a pesar de este inconveniente, existen argumentos muy sólidos a favor de esta dependencia:

1. Es más fácil convencer a los individuos y organizaciones para actualizar un complemento más que un navegador completo.

2. La gente está muy familiarizada con *Flash Player*, está instalado en más del 98 por ciento de los equipos conectados a Internet porque la instalación es muy sencilla e intuitiva. Además *Flash Player* ofrece de forma única una experiencia de usuario coherente y accesible desde múltiples exploradores y plataformas.

3. El proceso de actualización es todavía más sencillo que la instalación.

4. Actuales versiones de *Flash* ya se publican al mismo tiempo para las tres plataformas principales (excepto para Linux de 64 bits).

Tradicionalmente, el desarrollo de contenido *Flash* se ha apoyado en la herramienta autorizada, *Flash 8 Professional*, aunque ahora se puede llevar a cabo de otras formas:

- Utilizar únicamente la aplicación *Flash CS3* para crear y organizar contenido, y añadir ActionScript a ese framework.
- Utilizar un IDE con todas las funciones necesarias como por ejemplo *Adobe Flex Builder*. Con este software se acelera de manera espectacular el desarrollo de aplicaciones *Flex* con un elevado uso de ActionScript.
- Utilizar la aplicación gratuita *Adobe Flex SDK* y crear aplicaciones *Flex* desde línea de comandos.

Las aplicaciones Flex son compiladas por la máquina virtual de *Flash* a archivos SWF *just in time (JIT)*. En este PFC se ha elegido la segunda opción que se explica con más detalle a continuación.

Flex

Adobe Flex (hasta 2005 Macromedia Flex) constituye un marco de trabajo de código abierto gratuito altamente productivo para la creación y el mantenimiento de aplicaciones RIA basadas en la plataforma *Flash*; estas aplicaciones se implantan coherentemente en los principales navegadores y sistemas operativos ya que utilizan el software *Adobe Flash Player*.

Flash fue pensado para crear interfaces de usuario y ésto le proporciona un atractivo adicional a *Flex*, que en realidad es un DSL (*Domain-Specific Language*) [1]. Sin embargo, Flex permite crear aplicaciones sofisticadas de todos los tipos ya que incorpora una biblioteca de componentes muy completa. Otras características de Flex son:

- Ofrece un modelo de programación robusto que resultar familiar a los desarrolladores con conocimientos básicos sobre programación orientada a objetos.
- Ofrece una sintaxis de definición formal de clases.
- Ofrece una estructura formal de paquetes y herencia.
- Tiene un tipado de variables, parámetros y valores de retorno.
- Ofrece funciones **get/set** implícitas que usan las palabras reservadas **get** y **set**.
- Introduce el concepto de clases cerradas.
  Una clase cerrada posee únicamente el conjunto fijo de propiedades y métodos definidos durante la compilación; no es posible añadir propiedades y métodos adicionales a menos que se conviertan en dinámicas (utilizando la palabra reservada *dynamic*). Cerrarlas permite

[1] La mayoría de las soluciones pensadas para el diseño de interfaces gráficas han sido lenguajes con librerías UI)

realizar una comprobación más estricta en tiempo de compilación, lo que aporta una mayor solidez a los programas. Todas las clases de ActionScript 3.0 están cerradas de forma predeterminada.

Flex incluye un lenguaje declarativo (MXML) para definir las interfaces de usuario y un lenguaje de programación llamado ActionScript, que es un superconjunto de ECMAScript con características adicionales como la comprobación de tipos estáticos. ECMAScript es equivalente a JavaScript y por tanto se pueden aprovechar los conocimientos en este lenguaje.

Figura 2.2: Arquitectura aplicación FLEX

Lenguaje MXML

MXML (*Multimedia eXtensible Markup Language*) es un lenguaje descriptivo basado en XML y desarrollado inicialmente por Macromedia hasta el 2005 para Flex. Este lenguaje se utiliza para describir el aspecto y comportamiento de la interfaz de usuario. MXML tiene una mayor estructura en base a etiquetas, similar a HTML, pero con una sintaxis menos ambigua. Proporciona una gran variedad e inclusive permite extender etiquetas y crear sus propios componentes. La mayoría de los tags MXML corresponden a clases de ActionScript 3 o propiedades de dichas clases.

Lenguaje ActionScript 3.0

ActionScript es un lenguaje de programación orientado a objetos (OOP), utilizado especialmente en aplicaciones web realizadas en el entorno Adobe Flash para implementar la lógica de cliente. El lenguaje está basado en especificaciones del estándar de industria ECMA-262, un estándar para JavaScript, de ahí que ActionScript se parezca tanto a JavaScript. La versión más extendida actualmente es ActionScript 3.0.

Flex Builder

Adobe Flex Builder es un entorno integrado de desarrollo basado en Eclipse que permite crear aplicaciones Flex (o aplicaciones Flash basadas en AS3) de una forma más productiva. Permite crear prototipos para las interfaces gráficas rápidamente, facilita la vida a la hora de programar, da pistas sobre el código y muchas veces lo genera automáticamente con atajos de teclado, etc. Por otro lado, proporciona herramientas de depuración y permite gestionar la memoria(*memory profiling*) mostrando en tiempo de ejecución cuántas instancias de cada objeto existen;ésto es muy útil para optimizar la aplicación y detectar fallos de rendimiento.

Los requisitos para utilizar *Adobe Flex Builder* son:

1. Sistema operativo Windows 2000, Windows XP o Windows 2003.
2. Mínimo espacio libre en disco de 300 MB.
3. Java Runtime Environment en versión 1.5.x (o superior).
4. Eclipse versión 3.3.x.

Para el mundo Linux es muy importante contar con una herramienta como ésta, del prestigio de Adobe. La versión alpha de *Flex Builder* para Linux soporta la mayoría de las características de Flex Builder 3, incluyendo facilidades para la creación del proyecto, coloreado y sugerencias de código, búsqueda de referencias, etc. Esta versión de Flex Builder para Linux apoya las siguientes distribuciones:

1. SuSE Linux Enterprise Server 10 (x86 32-bit).
2. RedHat Enterprise Linux WS 4 (x86 de 32-bit).
3. Ubuntu 7.0.4 (Feisty) (x86 de 32-bit).
4. Ubuntu 7.10 (Gutsy Gibbon) (x86 de 32-bit).
5. Fedora Core 8 (x86 de 32-bit).

Se puede combinar el lenguaje Java (Python, Ruby, C ++, etc) en el lado del servidor (back-end) y Flex en el lado del cliente (para construir una interfaz interactiva). La comunicación entre una interfaz de usuario de *Flex* y un servidor o aplicación local sólo puede ser inicada desde la interfaz gráfica. Ésto es así por cuestiones de seguridad (sería equivalente a tener un puerto abierto en un equipo).

Comunicación cliente-servidor

La tecnología Flex puede emplear tres métodos para la comunicación cliente-servidor:

- *HTTP Services*.
  Permite hacer llamadas a servicios ASP, PHP, JSP, ColdFusion, etc, utilizando el protocolo HTTP (ver Figura 2.3).

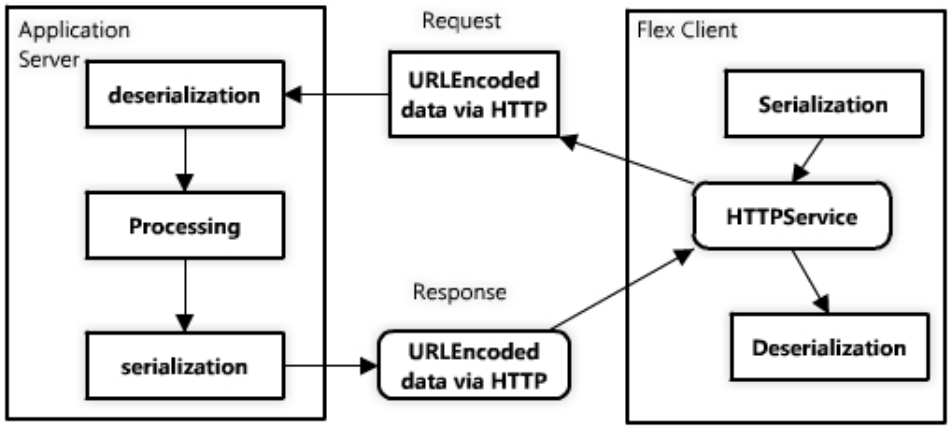

Figura 2.3: HTTP Services

  Ésto es equivalente al objeto *HTTPRequest* de Javascript (Ajax).

- *Web Services*.
  Son servicios web basados en SOAP de los que se puede obtener información específica utilizando los métodos descritos en el servicio web. La respuesta de un WebService es un XML formateado bajo estándares ya establecidos (ver Figura 2.4).
  HTTP Services y *Web Services* forman parte de los servicios RPC (*Remote Procedure Call*).

- Protocolo AMF3.
  Para resolver el problema de comunicación y de interoperabilidad de las aplicaciones *Flash* se creó el protocolo AMF (*ActionScript Message Format*). Dicho protocolo ayuda a la transmisión de los denominados objetos remotos entre el *front-end* y el *back-end*. En la Figura 2.5 se muestra un esquema de su funcionamiento.

Para el protocolo AMF existe una solución comercial denominada LCDS (*Adobe LiveCycle Data Services*) y otras soluciones alternativas gratuitas como *BlazeDS* y *GraniteDS*. Estas últimas ofrecen sólo un subconjunto de las funcionalidades de LCDS aunque todas utilizan la serialización estándar de AMF3, que impide que ciertas propiedades (las *no-transient* y *static public*) puedan ser serializadas.

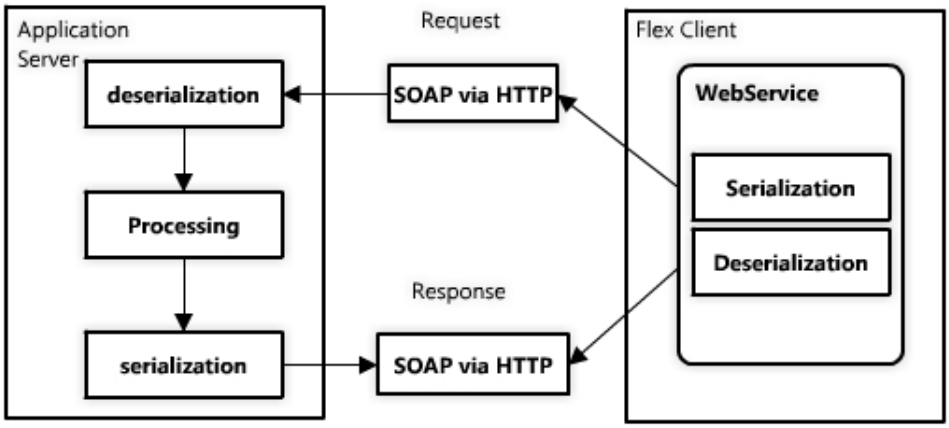

Figura 2.4: Web Services

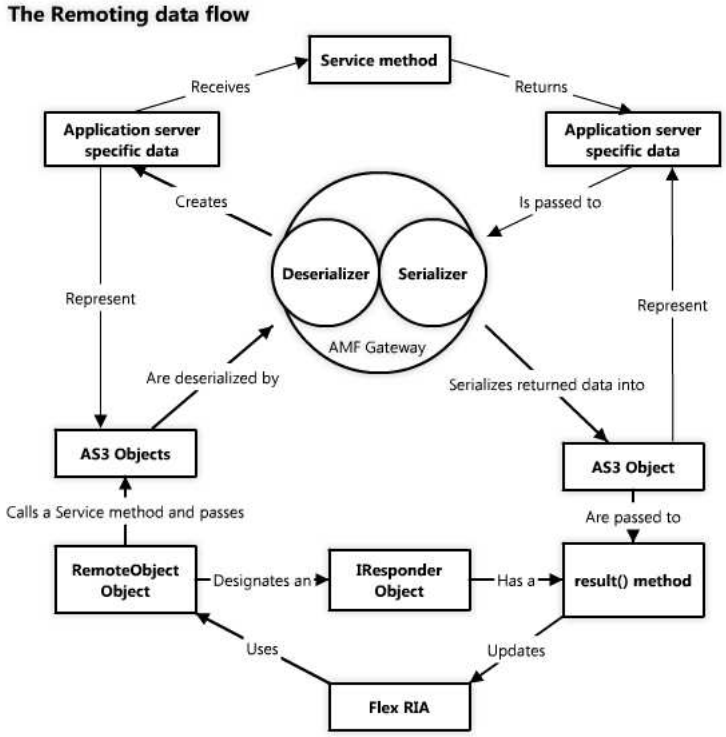

Figura 2.5: AMF

GraniteDS

GraniteDS implementa las factorías de servicio necesarias para: EJB 3 (beans de sesión), *Spring*, POJO (clases Java simples) sin afectar a su rendimiento, etc. Su arquitectura viene representada en la Figura 2.6.

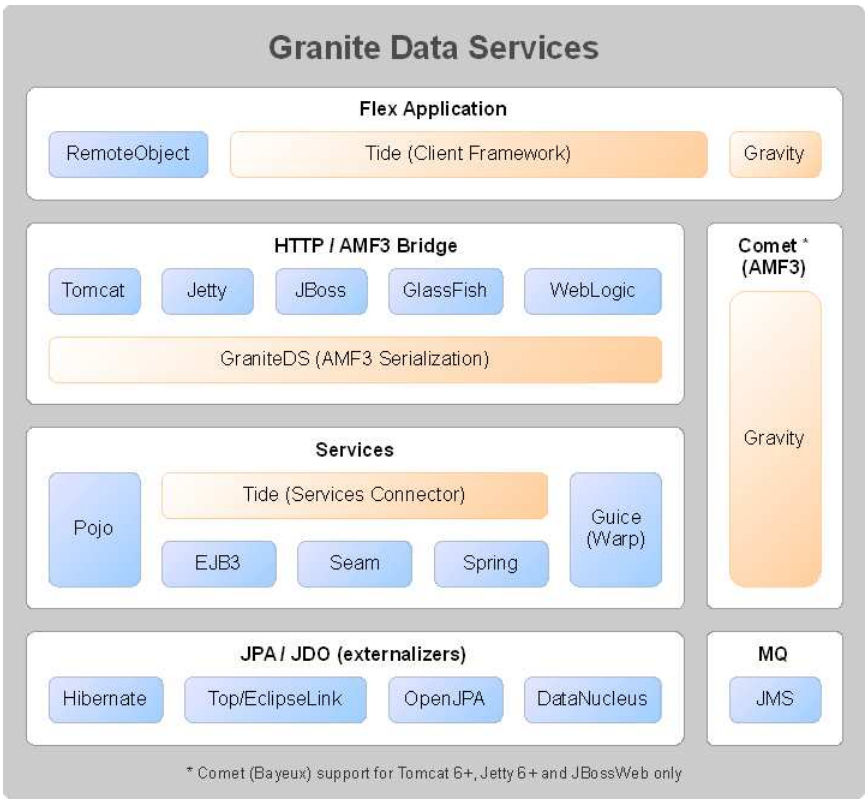

Figura 2.6: Arquitectura Granite DS

GraniteDS hace uso de *externalizers*, que requieren beans ActionScript que implementen la interfaz *flash.utils.IExternalizable*. La escritura y el mantenimiento de estos beans ActionScript es tedioso y una fuente de errores; para resolverlos y acelerar el desarrollo de aplicaciones Flex/J2EE *GraniteDS* viene con un generador de código ActionScript denominado GAS3. Un problema común con los generadores de código es la pérdida potencial de modificaciones manuales realizados en los archivos generados. El archivo generado no es capaz de reflejar las modificaciones introducidas en el modelo; se genera una vez y sólo una vez, no se regenera cada vez que se modifica el modelo.

2.2. MVC

2.2.1. Arquitectura MVC

En la fase de diseño se toman decisiones básicas sobre cómo escribir código reutilizable, cómo acceder a los recursos, etc. En la fase de diseño también se definen los entornos de desarrollo y despliegue, incluyendo la estructura de directorios de la aplicación. Pero con estas decisiones no basta, todavía faltan aspectos de arquitectura por decidir.

Un punto de partida común en la fase de diseño consiste en identificar uno o más patrones adecuados a la aplicación. Un patrón de diseño describe una solución a un problema de programació (aunque puede dar una idea de cómo acercarse al diseño, no necesariamente define cómo escribir el código para dicha solución). Se han catalogado y documentado muchos tipos de patrones de diseño. Por ejemplo, el modelo de diseño funcional especifica que cada módulo de la aplicación realiza una sola acción, sin impacto alguno en los otros módulos de la aplicación; aunque no se especifica qué es un módulo, normalmente corresponde a una clase o método.

El patrón de diseño MVC (*Model-View-Controller*) se originó en el lenguaje Smalltalk y se ha utilizado ampliamente durante años. Aunque los principios básicos del modelo son fáciles de entender, sus detalles son lo suficientemente complejos como para fomentar una enorme cantidad de implementaciones contradictorias. Su objetivo es aumentar la reutilización de los componentes y mejorar la mantenibilidad del sistema mediante la creación de componentes con un alcance limitado y bien definido. Está pensado para reducir el esfuerzo de programación necesario en la implementación de varios sistemas y modificar los componentes sin afectar a todo el sistema. Separar el código en clases de modelo, vista y controlador conlleva además los siguientes beneficios:

- Permite múltiples representaciones (vistas) de la misma información (modelo).
- Permite que las interfaces de usuario (vistas) sean fáciles de añadir, modificar, eliminar tanto en tiempo de compilación como en tiempo de ejecución.
- Permite que el controlador sea fácil de cambiar tanto en tiempo de compilación como en tiempo de ejecución.
- Promueve la reutilización.
- Permite que varios desarrolladores puedan actualizar la interfaz, la lógica o la vista simultáneamente, sin afectar al código fuente de otros.
- Ayuda a que los desarrolladores puedan centrarse en un solo aspecto de la aplicación.

Utilizando la arquitectura MVC (ver Figura 2.7), un sistema se puede dividir en tres categorías de componentes:

- Componentes del modelo: son los datos procesados por la aplicación que vienen en forma de fichero de texto, base de datos, documento XML, etc.
- Componentes de negocio: son los programas, scripts encargados de la interconexión de datos en la aplicación, gestionando la lógica de negocio. El controlador no tiene por qué tener conocimiento de la vista o el modelo.
- Componentes de vista: definen la interfaz de usuario de la aplicación.

Se puede considerar una aplicación Flex como parte de la vista en una arquitectura MVC distribuida, pero en realidad una aplicación Flex tiene su propia visión de los componentes de vista, negocio y modelo.

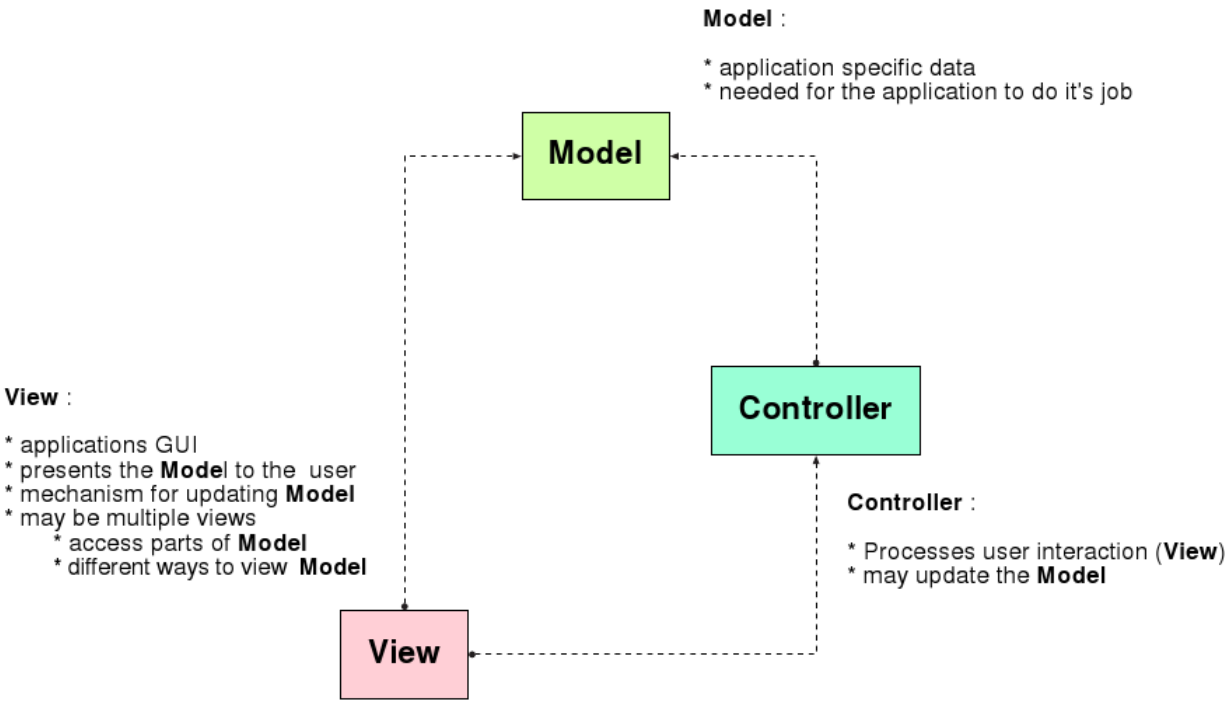

Figura 2.7: Arquitectura Model-View-Controller

2.2.2. Comunicación en MVC

Aunque modelo, vista, y controlador son componentes separados, deben comunicarse regularmente:

- El modelo debe enviar notificaciones de cambios de estado a la vista.

- La vista debe registrar el controlador para recibir eventos de la interfaz de usuario.
- El controlador debe actualizar el modelo y actualizar la vista en respuesta a la entrada del usuario.

Para facilitar esta comunicación, cada objeto debe almacenar una referencia a los otros con los que interactúa tal y como se muestra en la Figura 2.8:

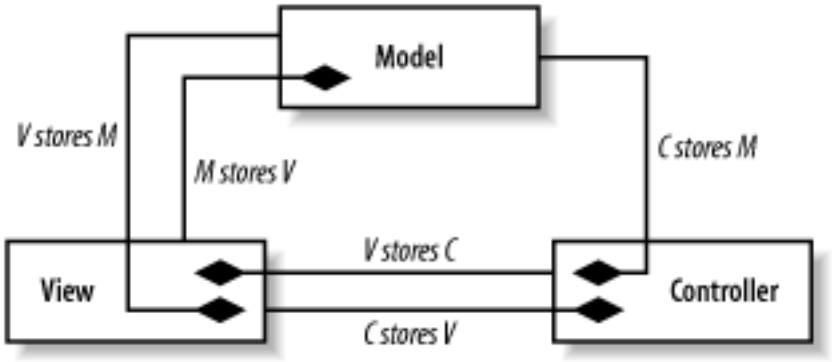

Figura 2.8: Arquitectura Model View Controller - referencias entre componentes

El punto de partida suele ser la entrada del usuario. Sin embargo, otra parte del programa también podría iniciar el ciclo de la modificación del modelo directamente. En cualquier caso, en la Figura 2.9 se muestra el ciclo de comunicación MVC.

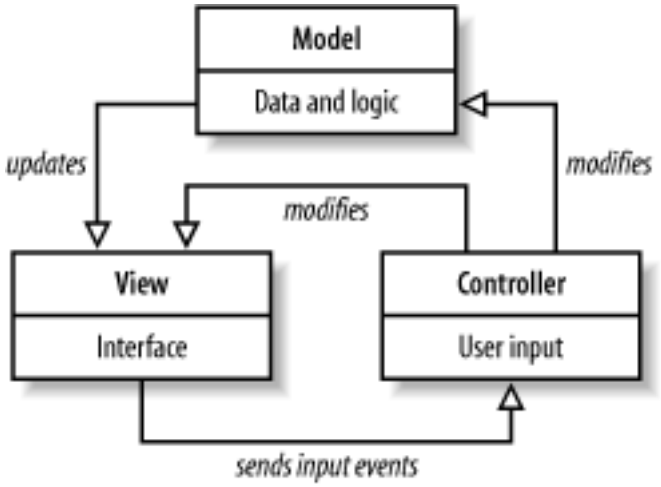

Figura 2.9: Arquitectura Model-View-Controller - comunicación entre componentes

En la figura anterior se muestra el caso más simple de la arquitectura MVC, con un modelo único, una vista única y un solo controlador. Sin embargo, no es raro que existan dos o más vistas de un modelo único, pero cada vista solo tiene una instancia de controlador y viceversa.

2.3. APIs Google

En este PFC se ha utilizado el API de Google Maps para *Flash* para integrar muchas de sus características en la aplicación web desarrollada. Las utilidades proporcionadas permiten manipular y añadir contenido a los mapas mediante diversos servicios.

2.3.1. API Google Maps

Google Maps es el nombre de un servicio gratuito de Google. Se trata de un servidor de aplicaciones de mapas en web que ofrece imágenes de mapas desplazables, fotos satelitales de todo el mundo e incluso la ruta entre diferentes ubicaciones. El usuario puede controlar el mapa con el mouse o las teclas de dirección para moverse a la ubicación que se desee o hacer zoom.

Existen dos versiones del API de Google Maps, una ya existente para Javascript y otra para *Flash* (ésta última es una alternativa completamente independiente de la primera). Para combinar contenido Flash con Google Maps se debe incluir en la aplicación un archivo SWC que viene en el SDK de Google Maps para *Flash*. Compilar una aplicación en estas condiciones garantiza que se puedan utilizar todas las funciones públicas de la biblioteca del tiempo de ejecución del API de Google Maps ya que el archivo SWC contiene las interfaces correspondientes. El hecho de que la información se recupere desde los servidores de Google cada vez que un cliente carga una aplicación permite realizar mejoras, solucionar errores y hacer modificaciones en las funciones de la biblioteca principal sin tener que recompilar la aplicación.

El API de Google Maps para *Flash*, al igual que el API de Google Maps para Javascript, es un servicio beta que requiere el uso de una clave de desarrollador disponible de forma gratuita. Dicha clave, que es válida para un único **dominio** y que estará vinculada a la cuenta de Google del usuario, debe ser especificada en alguna de las siguientes ubicaciones:

1. En la declaración MXML de las aplicaciones Flex.

2. En el código ActionScript (compilado en el archivo SWF resultante) de las aplicaciones *Flash*.

3. En el elemento DOM contenedor de la página web.

Aunque no se ha incluido en este PFC sí se estudió la posibilidad de utilizar el API de datos de Google para que las modificaciones realizadas en la aplicación pudieran ser vistas directamente en la pgina web de Google. Éste es el objetivo de Google: organizar la información mundial y hacerla universalmente accesible y útil.

2.3.2. API Google Data

Los APIs de datos de Google (Google Data) constituyen un mecanismo para leer e introducir datos y así actualizar los contenidos de Google. Disponen un sistema de publicación de *web feeds* que incluye el protocolo **Atom** y algunas extensiones para gestionar consultas.

Un *web feed* es un documento (normalmente basado en XML) que contiene entradas (artículos de texto completo, etc) y/o enlaces a contenidos de cualquier sitio web, junto con los correspondientes metadatos.

Atom incluye un lenguaje XML utilizado para *web feeds* y un protocolo HTTP (*Atom Publishing Protocol*) basado en la creación y actualización de los recursos web. El formato Atom fue desarrollado como una alternativa a RSS por las limitaciones de este último y fue publicado como un estándar de IETF en el RFC 4287. Un ejemplo de documento en formato Atom es el siguiente:

```
1  <?xml version="1.0" encoding="utf-8"?>
2
3  <feed xmlns="http://www.w3.org/2005/Atom">
4
5          <title>Example Feed</title>
6          <subtitle>A subtitle.</subtitle>
7          <link href="http://example.org/feed/" rel="self" />
8          <link href="http://example.org/" />
9          <id>urn:uuid:60a76c80-d399-11d9-b91C-0003939e0af6</id>
10         <updated>2003-12-13T18:30:02Z</updated>
11         <author>
12           <name>John Doe</name>
13           <email>johndoe@example.com</email>
14         </author>
15
16         <entry>
17           <title>Atom-Powered Robots Run Amok</title>
18           <link href="http://example.org/2003/12/13/atom03" />
19           <link rel="alternate"
20                 type="text/html"
21                      href="http://example.org/2003/12/13/atom03.html"/>
22           <link rel="edit"
23                 href="http://example.org/2003/12/13/atom03/edit"/>
24           <id>urn:uuid:1225c695-cfb8-4ebb-aaaa-80da344efa6a</id>
25           <updated>2003-12-13T18:30:02Z</updated>
26           <summary>Some text.</summary>
27         </entry>
28
29 </feed>
```

Por su parte, el protocolo de publicación Atom fue publicado como estándar de IETF en el RFC 5023.

API Maps Data

El API Maps Data permite a las aplicaciones cliente crear, eliminar o actualizar mapas ya existentes empleando los **feeds** del API Google Data y un modelo de datos de características (marcas de posición, líneas y formas) y mapas (conjuntos de características). Estos **feeds** proporcionan las direcciones URL que utiliza el estándar Atom; de forma programática se pueden crear, modificar y eliminar datos mediante peticiones HTTP.

Las ventajas de utilizar el API Maps Data son:

- Escalabilidad
  No hay que preocuparse por el mantenimiento de un almacén de datos para crear un *mashup* de Google Maps; el alojamiento y ancho de banda son gratuitos.
- Accesibilidad
  Con las bibliotecas de cliente el acceso a datos geográficos almacenados es posible desde cualquier dispositivo.
- Velocidad
  Obtener los datos geográficos requiere indexación en tiempo real. Con el API Maps Data estos datos pueden ser instantáneamente indexados y realizar búsquedas en Google Maps.
- Presentación
  La renderización de los datos geográficos es más rápida con las herramientas adecuadas.

El API Maps Data proporciona los siguientes **feeds**:

1. Map.
2. Feature.

El **feed** *Map* es un **feed** de mapas creados por el usuario. Cada uno de ellos es devuelto como una entrada independiente y las operaciones permitidas sobre cada uno de ellos son:

1. query (GET http://maps.google.com/maps/feeds/maps/userID/full/mapID)
   Los parmetros soportados en esta operacin son: *max-results*, *showdeleted*, *start-index*.
2. insert (POST http://maps.google.com/maps/feeds/maps/userID/full)
3. update (PUT http://maps.google.com/maps/feeds/maps/userID/full/mapID)
4. delete (DELETE http://maps.google.com/maps/feeds/maps/userID/full/mapID)
5. batch (POST http://maps.google.com/maps/feeds/maps/userID/full/batch)

La URI completa de este feed es http://maps.google.com/maps/feeds/maps/default/full

El **feed** *Feature* es un **feed** de las características del mapa. Cada una de ellas es devuelta como una entrada independiente y las operaciones permitidas sobre cada una de ellas son:

1. query (GET http://maps.google.com/maps/feeds/features/userID/mapID/full/featureID)
   Los parmetros soportados en esta operacin son: *max-results*, *showdeleted*, *start-index*.
2. insert (POST http://maps.google.com/maps/feeds/features/userID/mapID/full)
3. update (PUT http://maps.google.com/maps/feeds/festures/userID/mapID/full/featureID)
4. delete (DELETE http://maps.google.com/maps/feeds/festures/userID/mapID/full/featureID)
5. batch (POST http://maps.google.com/maps/feeds/festures/userID/mapID/full/batch)

La URI completa de este feed es http://maps.google.com/maps/feeds/features/userID/mapID/full

Capítulo 3

Android

Android es un software (de código abierto) desarrollado por Google y por la *Open Handset Alliance* para dispositivos móviles. Incluye un sistema operativo basado en Linux, por ello todos los servicios base (gestión de drivers, memoria, seguridad, etc) están incluidos en dicho sistema operativo.

Aunque Android se puede instalar en cualquier terminal, está orientado a los *SmartPhone* ya que todas las aplicaciones de Google son accesibles a través de Internet y su uso es mayor en estos dispositivos. De todos modos Android ha optimizado al máximo el uso de recursos del móvil para funcionar en terminales poco potentes.

3.1. Características principales

- Incluye un SDK que proporciona las herramientas necesarias para desarrollar aplicaciones en la plataforma Android utilizando el lenguaje de programación Java.
- Viene con un motor de navegación integrado (*Webkit*), el mismo que utilizan los *Mac* o los *IPhone*.
- Dispone de un framework que permite la reutilización de componentes y gráficos.
- Utiliza SQLite (que viene incluido en el SDK) para el almacenamiento estructurado de datos aunque también se pueden utilizar otras bases de datos como Perst.
- Soporta formatos comunes de audio, vídeo e imagen (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF).
- Soporta telefonía GSM (dependiente del hardware), Bluetooth, EDGE, 3G, y WiFi (dependiente del hardware).
- Dispone de cámara, GPS y brújula.

- Proporciona un entorno de desarrollo muy variado, incluyendo un emulador de dispositivos, herramientas para la depuración y un plug-in para el IDE Eclipse.

- Viene con un conjunto de aplicaciones básicas (cliente de correo electrónico, calendario, mapas, navegador, etc).

- Tiene una máquina virtual (*Dalvik*) optimizada para requerir poca memoria y poder ejecutar varias instancias simultáneamente sin que el dispositivo se ralentice. Esto es importante porque en Android cada aplicación se ejecuta en un proceso separado que utiliza su propia instancia de la máquina virtual.
  Los ejecutables tienen la extensión .dex, una versión optimizada de los .class. Esta conversión podría ser una estrategia de Google para evitar conflictos con Sun por la licencia de la máquina virtual.

La estructura de los proyectos Android es la siguiente:

- Directorio src: contiene la clase principal y el archivo R.java.
  La clase R.java es actualizada automáticamente con cada modificación; es un índice a todos los recursos declarados en el proyecto para hacer más rápida la búsqueda de referencias.

- AndroidManifiest.xml: declara todas las activities e intents de la aplicación.
  En la creación del proyecto se puede definir la activity que representará la pantalla inicial de la aplicación.

- Directorio res: contiene los siguientes subdirectorios:

  1. Directorio drawable: contiene las imágenes (en formato .PNG) que utiliza la aplicación.
  2. Directorio layout: contiene ficheros donde se definene los elementos que conforman la interfaz gráfica de la aplicación.
     En Android las UIs se pueden construir de dos maneras:
     - Invocando métodos de las clases del API, lo que complica cualquier modificación sobre una UI.
     - empleando archivos XML donde cada etiqueta del árbol es un elemento del tipo View representado dentro de la pantalla. Así, resulta mucho más rápido y sencillo crear una UI. Este modelo alternativo se denomina *XML-base layout*.
       Este modelo está basado en el diseño de páginas Web, donde se separa la presentación de la lógica.
  3. Directorio values: contiene el archivo strings.xml que define las cadenas de texto utilizadas.

3.2. Arquitectura Android

La arquitectura de Android (ver Figura 3.1) está formada por capas de software donde cada una puede utilizar los servicios de la capa inferior. Empezando por la capa inferior está el conjunto de drivers basados en Linux, esta parte no es pública. Un nivel más arriba está un conjunto de librerías que no son accesibles directamente sino a través del nivel superior a ésta, el Framework de aplicaciones, que junto a la capa de aplicaciones son totalmente públicas y los usuarios pueden acceder libremente.

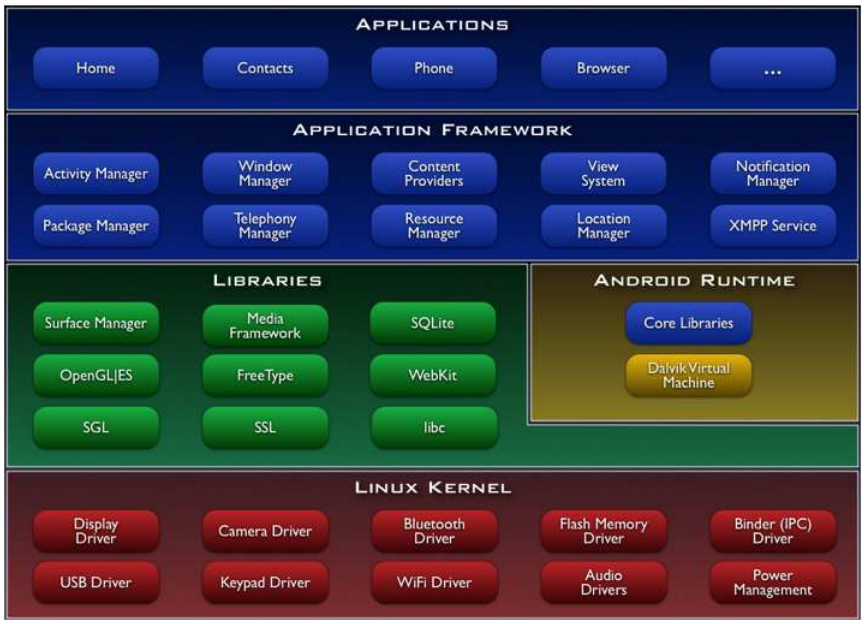

Figura 3.1: Arquitectura Android

Uno de los componentes principales de la arquitectura de Android es el módulo *Location Manager* (ver Figura 3.2). Las clases que implementan este componente permiten a las aplicaciones Android obtener actualizaciones periódicas de la situación geográfica del dispositivo. También es posible disparar una aplicación específica cuando el dispositivo entra en la proximidad de un lugar geográfico determinado.

3.2.1. Técnicas localización

En un sistema de localización se deben diferenciar dos términos: *posición*, que se refiere a la posición exacta del dispositivo, y *localización*, que se refiere a información secundaria derivada de la posición del dispositivo.

Saber dónde está un dispositivo móvil, y por tanto, dónde está el usuario, es la base de todas las aplicaciones que tienen como objetivo ofrecer servicios basados en recursos geolocal-

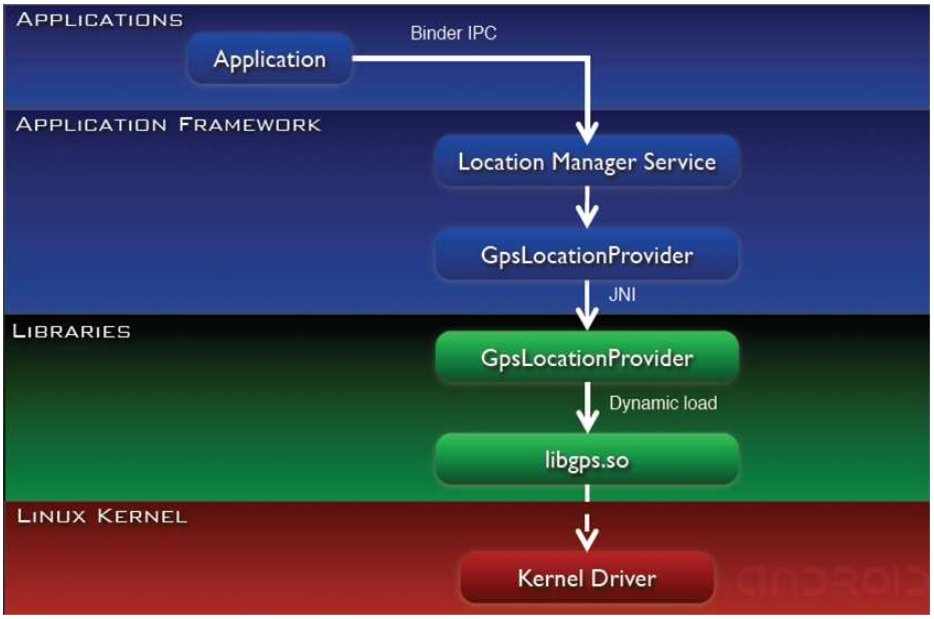

Figura 3.2: Componente Location Manager

izados. Esto permite personalizar la información que solicita el usuario, haciendo más cómoda su experiencia de uso y aunando en lo que realmente necesita y le interesa. Cada vez surgen más aplicaciones de este tipo para ejecutarse en dispositivos móviles de nueva generación, demostrando que es una vía que promete un gran futuro en el campo de los servicios a clientes.

A las técnicas de localización más conocidas (por ejemplo GPS) se han añadido otras más recientes y generalmente desconocidas por el público general, pero no por ello menos efectivas, que, incluso, complementan a las tradicionales. Hoy en día es fácil encontrar en los dispositivos móviles al menos una manera de interactuar con alguna de ellas, lo que hace que los servicios geolocalizados sean usables por un gran público. A continuación se exponen dos de las más populares: GSM, GPS.

GPS

GPS permite calcular la posición de un objeto en un espacio de coordenadas de tres dimensiones, a partir de las distancias del objeto a tres satélites (como mínimo) de localización conocida. Este sistema fue diseñado para proveer a las unidades militares de Estados Unidos de un sistema que les permitiera conocer su posición en todo momento y en cualquier lugar del mundo. En los últimos años el precio de los terminales que lo soportan se ha reducido considerablemente, lo que ha permitido su desarrollo e integración en dispositivos inalámbricos, apoyando la oferta de servicios de localización.

La navegación por satélite permite la emisión de señales desde satélites indicando un espacio

y tiempo extremadamente precisos. Gracias a un pequeño receptor individual esta tecnología permite conocer la localización de cualquier dispositivo. Hoy día existen dos redes de satélites de radionavegación: uno americano, GPS, y otro ruso, *Glonass*.

El sistema de posicionamiento GPS es adecuado para entornos exteriores porque existe visión directa de los satélites. La distancia entre el receptor GPS y un satélite se mide multiplicando el tiempo de vuelo de la señal emitida desde el satélite por su velocidad de propagación. El tiempo de vuelo es extremadamente pequeño y se necesitan dispositivos de cronometraje extremadamente precisos para medirlo con exactitud; pequeños errores de cronometraje pueden resultar en grandes errores de posición. Para medir este tiempo es necesario que los relojes de los satélites y de los receptores estén sincronizados, pues deben generar simultáneamente el mismo código pseudo-aleatorio. El receptor calcula la distancia realizando un desplazamiento temporal de su código pseudo-aleatorio hasta lograr la coincidencia con el código recibido; este desplazamiento corresponde al tiempo de vuelo de la señal.

Por otro lado, las señales GPS son muy débiles y se hallan inmersas en el ruido de fondo inherente al planeta en la banda de radio. Los receptores GPS deben recibir correctamente la débil señal de los satélites, por lo que están pensados para utilizarse en el exterior y pueden no funcionar correctamente en el interior de un bosque tupido o entre montañas o edificios altos; existen otros sistemas más fiables para el caso de entornos cerrados.

El sistema GPS se descompone en tres segmentos básicos:

- Segmento Espacial: formado por 24 satélites de la constelación NAVSTAR, distribuidos en seis planos orbitales que forman una red que envuelve la esfera terrestre.
  Se utilizan las señales procedentes de los veinticuatro satélites para determinar por triangulación, la altitud, longitud y latitud de cualquier objeto en la superficie terrestre.

- Segmento Control: formado por varias estaciones de rastreo distribuidas en la superficie terrestre, que continuamente monitorizan a cada satélite analizando las señales emitidas y actualizando los datos de los elementos y mensajes de navegación, así como las correcciones de reloj de los satélites.

- Segmento Usuario: formado por los receptores pasivos situados en tierra que decodifican los mensajes que provienen de cada satélite visible. Combinando la información de todos los satélites, el receptor GPS es capaz de estimar su propia posición, es decir sus coordenadas de latitud y longitud [1] con una precisión aproximada de diez metros.

[1] La mayoría de receptores proporcionan los valores de estas coordenadas en unidades de grados y minutos, con respecto a un cero de referencia bien definido.

GSM

La información sobre localización de un usuario está presente en redes GSM, el sistema conoce la célula en la que se encuentra el terminal para establecer y mantener una comunicación. Los métodos que se han descrito en la Sección se pueden adaptar a redes de telefonía móvil celular; algunos son directamente configurables y otros necesitan que se efectúen modificaciones en la red o en el terminal de usuario.

En GSM el nivel de señal medido por el dispositivo móvil presenta una variabiliad espacial importante (en un rango de uno a diez metros); sin embargo, el nivel de señal en una posición fija apenas varía a lo largo del tiempo.

Los sistemas de localización basados en GSM presentan las siguientes ventajas:

1. La cobertura de GSM es total.

2. La amplia aceptación de los teléfonos móviles los hace ideales para la entrega de aplicaciones relacionadas con la computación ubicua.

3. El hardware existente de los teléfonos se puede utilizar sin necesidad de interfaces radio adicionales.

4. Pueden funcionar aunque falle la infraestructura eléctrica de un edificio.

5. GSM opera en una banda de frecuencia con licencia, por tanto no sufre interferencias de otros dispositivos transmitiendo a la misma frecuencia (microondas, etc).

6. La red GSM es reconfigurada con muy poca frecuencia; la estabilidad del entorno permite al sistema de localización operar durante un largo periodo antes ser recalibrado.

Las operadoras de telefonía móvil en Europa están ofreciendo servicios basados en localización mediante técnicas de identificación celular. Para ubicar cualquier dispositivo con mayores prestaciones, existe un método mejorado basado en identificación celular (*Enhanced Cell-ID*) [**?**]. Este método utiliza el parámetro de avance temporal (*timing advance*), que es una señal estimada por la estación base (BTS) cuando el terminal accede por el canal de control de acceso aleatorio (*Random Access Channel* - RACH). [2]. El parámetro TA indica al móvil el tiempo que debe adelantar su transmisión con respecto al reloj de trama. Así pues, el terminal inicia su transmisión en el instante T_0 - TA (T_0 el instante en el que teóricamente debía iniciar su transmisión). El centro del servicio de localización de móviles (*Serving Mobile Location Center*) calcula la posición a partir de la celda en la que se encuentra el terminal, y del parámetro TA.

[2] Así se puede llegar a compensar un retardo de 233μs, proporcionando un rango de medidas de 35 kilómetros

3.3. Componentes

Una aplicación Android puede hacer uso (con los permisos correspondientes) de elementos de otras aplicaciones. Para que ésto funcione el sistema debe ser capaz de iniciar un proceso de aplicación cuando se necesite una parte de ella. Por ello las aplicaciones Android no tienen un punto de entrada único sino que tienen componentes esenciales que el sistema puede crear y ejecutar, según sea necesario. Existen cuatro tipos de componentes:

- Activity
  Una activity se encarga de presentar una interfaz de usuario. Se implementa como una subclase de la clase base **Activity**.
  Una aplicación Android puede estar compuesta por una o varias activities independientes. Normalmente una de las activities será la primera que se le presente al usuario cuando se inicie la aplicación, pero todas deben ser declaradas en el fichero *AndroidManifiest*.
  A cada activity se le da una ventana por defecto para dibujar pero también puede hacer uso de ventanas adicionales, por ejemplo cuadros de diálogo emergentes. El contenido visual de la ventana es proporcionada por objetos derivados de la clase **View**; cada uno de ellos controla un espacio rectangular particular dentro de la ventana.
  Android incorpora una clase especial llamada **MapActivity** que adapta los conocidos mapas de Google para su edición y utilización en aplicaciones Android. Esta clase incorpora algunos métodos que permiten mostrar mapas en una posición geográfica determinada, dibujar elementos sobre él (control para el zoom de la pantalla, la vista por satélite), etc. Esta clase se puede utilizar de dos formas distintas:

  1. Recibiendo los parámetros X e Y de otro activity anterior.
  2. Estableciendo unas coordenadas fijas.

- Service
  Un service no muestra una interfaz de usuario sino que se ejecuta en segundo plano por un periodo indefinido de tiempo. Extiende la clase **Service**.
  Es posible establecer comunicación con un service (e iniciar el service si no está en ejecución) a través de la interfaz correspondiente.

- Broadcast receiver
  Un broadcast receiver es un componente que recibe y responde a mensajes broadcast. Una aplicación Android puede tener N broadcast receivers. Todos los receptores deben extender la clase **BroadcastReceiver**.
  Un broadcast receiver no muestra una interfaz de usuario aunque puede iniciar un activity en respuesta a la información recibida o utilizar la clase **NotificationManager** para conseguir la atención del usuario mediante parpadeo de la luz de fondo, vibración, etc.

- Content Provider
  Un content provider hace posible que las aplicaciones puedan recuperar y almacenar datos en el sistema de archivos, en una base de datos SQLite, etc. Sin embargo, las aplicaciones no invocan directamente los métodos del content provider sino los de la clase **ContentResolver**. Todos los content provider deben extender la clase **ContentProvider**.

Android garantiza que las solicitudes recibidas serán gestionadas por los componentes correspondientes (los arrancará si es necesario).

3.3.1. Activación de componentes

Los componentes descritos en el apartado anterior se activan mediante mensajes asíncronos. Dichos mensajes contienen información de la acción que se solicita, de los datos sobre los que actuar, etc. Para los broadcast receiver los intents designan la acción que se anuncia. Para activar por separado cada uno de los componentes se procede de la forma que se describe a continuacin.

- Activity
  Se invoca la función **startActivity** del objeto **Context** pasando como argumento un objeto **Intent**. Cada vez que se pasa a una activity nueva la anterior queda en background (ésta es la forma de cambiar de pantalla en una aplicación).

- Service
  Se invoca la función **startService** del objeto **Context** pasando como argumento un objeto **Intent**; internamente Android ejecuta la función **onStart** del objeto **Service**. Otra posibilidad es invocar la función **bindService** del objeto **Context** para para establecer una conexión entre el componente y el service, que recibe el objeto **Intent** en una llamada al método **onBind()** (si el service no se está ejecutando se arranca).

- Broadcast Receiver
  Se invoca cualquiera de las siguientes funciones sobre el objeto **Context**: **sendBroadcast**, **sendOrderedBroadcast** o **sendStickyBroadcast**; internamente Android ejecuta la función **onReceive** del objeto **BroadcastReceiver**.

3.4. Ciclo de vida de las aplicaciones

En una aplicación Android es el propio sistema el que determina el ciclo de vida de dicha aplicación, su duración en base a los módulos que se están ejecutando en cada momento, la memoria disponible, etc. Android clasifica los procesos de manera jerárquica según la importancia que tengan para el sistema:

- Proceso foreground
  Proceso (activity) en primer plano con el que interactúa el usuario. Este proceso sera el último que eliminará el sistema.

- Proceso visible
  Proceso (activity) que no está en primer plano pero sí es visible (por ejemplo, un cuadro de diálogo). Este proceso sólo se elimina en estados de memoria muy críticos al igual que los procesos foreground.

- Proceso background
  Proceso (activity) que no está visible en pantalla. Si se eliminara no tendría una repercusión directa en el usuario.

- Proceso vacío
  Proceso que no está asociado con ningún componente visualmente activo. El sistema los elimina con frecuencia y puede mantenerlos vivos si hay memoria suficiente para mejorar el tiempo de activación de otro componente de la aplicación.

Activities, intent receivers o services son componentes que tienen un impacto considerable en el tiempo de ejecución de las aplicaciones.

3.4.1. Ciclo de vida de las activities

El ciclo de vida de las activities es muy parecido al ciclo de vida de las aplicaciones aunque sólo trata con dichos componentes. A diferencia de las aplicaciones para hablar del ciclo de vida de una activity hay que hacer referencia a la pila de activities en ejecución. Cuando se pasa de una activity a otra, la nueva se coloca en la parte superior de la pila mientras que la activity anterior queda en background.

Las activities tienen cuatro estados según su interacción con el usuario:

- Activa o en ejecución
  Es el estado de la última activity creada por la aplicación y será la última que el sistema intente eliminar.

- En pausa
  Es el estado de una activity que deja de ser la principal pero todavía queda visible en pantalla.

- Parada
  A diferencia del estado anterior la activity no es visible en pantalla. El sistema la eliminará frecuentemente para liberar memoria.

- Eliminada
  Una activity se elimina cuando el sistema necesita memoria o cuando es ordenado desde la aplicación.

Para manejar el estado de las activities la clase **Activity** incluye los siguientes métodos: **onCreate()**, **onDestroy()**, **onPause()**, **onStop()**, **onFrezze()**, **onResume()**, **onRestart()** y **finish()**.

El ciclo de vida de una aplicación está relacionado con el ciclo de vida de una activity. Android considera imprescindibles aquellos procesos con los que interacciona el usuario e intenta que no se vean afectados por los estados críticos de memoria.

A continuación se describen algunas de las herramientas incluidas en el SDK de Android.

3.5. ADT (*Android Development Tools*)

ADT permite crear y depurar aplicaciones de Android fácilmente. Algunas de su características son:

1. Proporciona acceso a otras herramientas de desarrollo Android desde el IDE Eclipse. Por ejemplo, ADT permite explotar las capacidades de la herramienta DDMS.
2. Proporciona un asistente para cada nuevo proyecto, que ayuda a crear rápidamente y todos los archivos básicos necesarios para una aplicación Android.
3. Automatiza y simplifica el proceso de construcción de una aplicación Android.
4. Proporciona un editor de código de Android que ayuda a escribir código XML válido para el fichero Android.MANIFEST y los ficheros de recursos.
5. Permite exportar el proyecto a un APK firmado que puede ser distribuido a otros usuarios.

3.6. AVD (*Android Virtual Devices*)

Los AVDs permiten configurar el emulador para ajustarse tanto como sea posible al dispositivo móvil real. Cada AVD se compone de:

1. Un perfil hardware
   Se pueden establecer tantas opciones como sean necesarias para definir las características hardware del dispositivo real. Por ejemplo, se puede definir si el dispositivo tiene una cámara, si utiliza un teclado QWERTY, etc.

2. Una asignación a una imagen del sistema
   Se puede definir qué versión de la plataforma Android se ejecutará en el emulador.

3. Otras opciones
   Se pueden controlar las dimensiones de la pantalla, apariencia, etc.

Se pueden crear tantos AVDs como sean necesarios según los tipos de dispositivos a modelar y las plataformas Android que utilice la aplicación. Para crear y administrar AVDs se utiliza la herramienta provista en el SDK de Android.

3.7. ADB (*Android Debug Bridge*)

ADB es una herramienta versátil que permite administrar el estado de una instancia del emulador o dispositivo Android. Se trata de un programa cliente-servidor que incluye tres componentes:

1. Un cliente, que se ejecuta en el equipo de desarrollo [3].
2. Un servidor que se ejecuta como proceso en segundo plano en el equipo de desarrollo.
3. Un servidor que gestiona la comunicación entre el cliente y el demonio adb que se ejecuta en el emulador o dispositivo.
4. Un demonio que se ejecuta como proceso background en cada emulador o instancia de dispositivo.

Antes de iniciar un cliente ADB se comprueba primero si existe un proceso servidor en ejecución. Si no lo hay se crea y se hace bind al puerto TCP 5037 para escuchar los comandos enviados desde los clientes ADB.

La aplicación Android se ha desarrollado con el IDE Eclipse (con el plugin ADT instalado), que proporciona una integración transparente con la herramienta ADB por lo que no es necesario acceder por línea de comandos.

3.8. DDMS (*Dalvik Debug Monitor Service*)

El SDK de Android incluye un emulador de dispositivo móvil que se ejecuta en el equipo de desarrollo y permite probar aplicaciones Android sin utilizar un dispositivo físico. El emulador imita todo el hardware y software típico de un dispositivo móvil normal aunque no puede recibir o realizar llamadas de/a un dispositivo real.

[3] Otras herramientas de Android como el plugin ADT-DDMS permiten crear clientes ABD

En la Figura 3.3 se presenta una captura de pantalla del emulador de Android integrado en el IDE Eclipse.

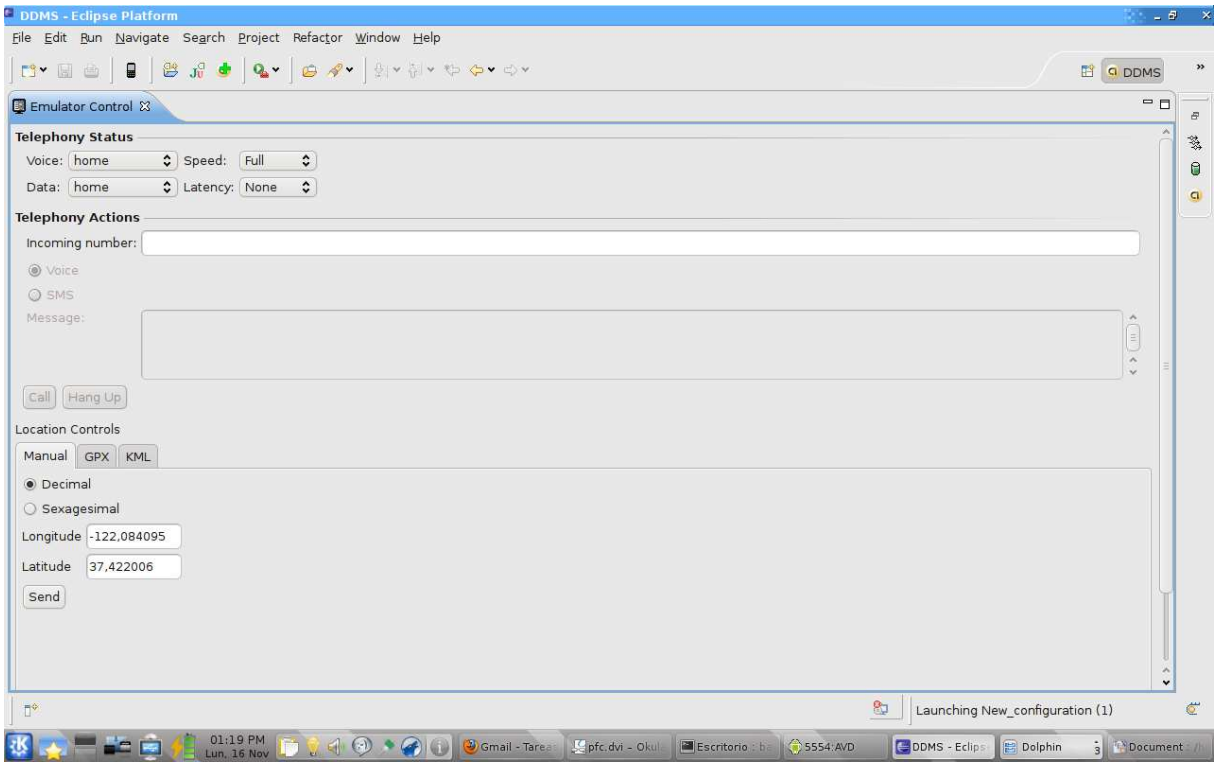

Figura 3.3: Emulator Control

El IDE Eclipse se puede conectar a las aplicaciones del emulador a través de la herramienta DDMS. En Android cada aplicación se ejecuta en su propio proceso, con su propia máquina virtual, por tanto DDMS utiliza diferentes puertos para conectarse al depurador de cada una de ellas. Inicialmente se crea un servicio de monitorización entre DDMS y ABD que permite detectar la conexión y desconexión de un dispositivo. DDMS es capaz de recuperar el identificador de proceso de cada máquina virtual y abrir una conexión con el depurador a través del demonio ABD del emulador.

En el monitor de depuración se muestran los emuladores/dispositivos encontrados con una lista de todas las máquinas virtuales en ejecución dentro de cada uno (ver Figura 3.4). Junto a cada máquina virtual aparece un puerto que conecta directamente al depurador.

Por otro lado, en la vista de hilos (ver Figura 3.5) se muestra una lista de subprocesos que se ejecutan en el proceso de la máquina virtual. Esta ficha contiene la siguiente información:

- ID
  Es el identificador asignado a la hebra de la VM (en Dalvik son números impares a partir de 3).

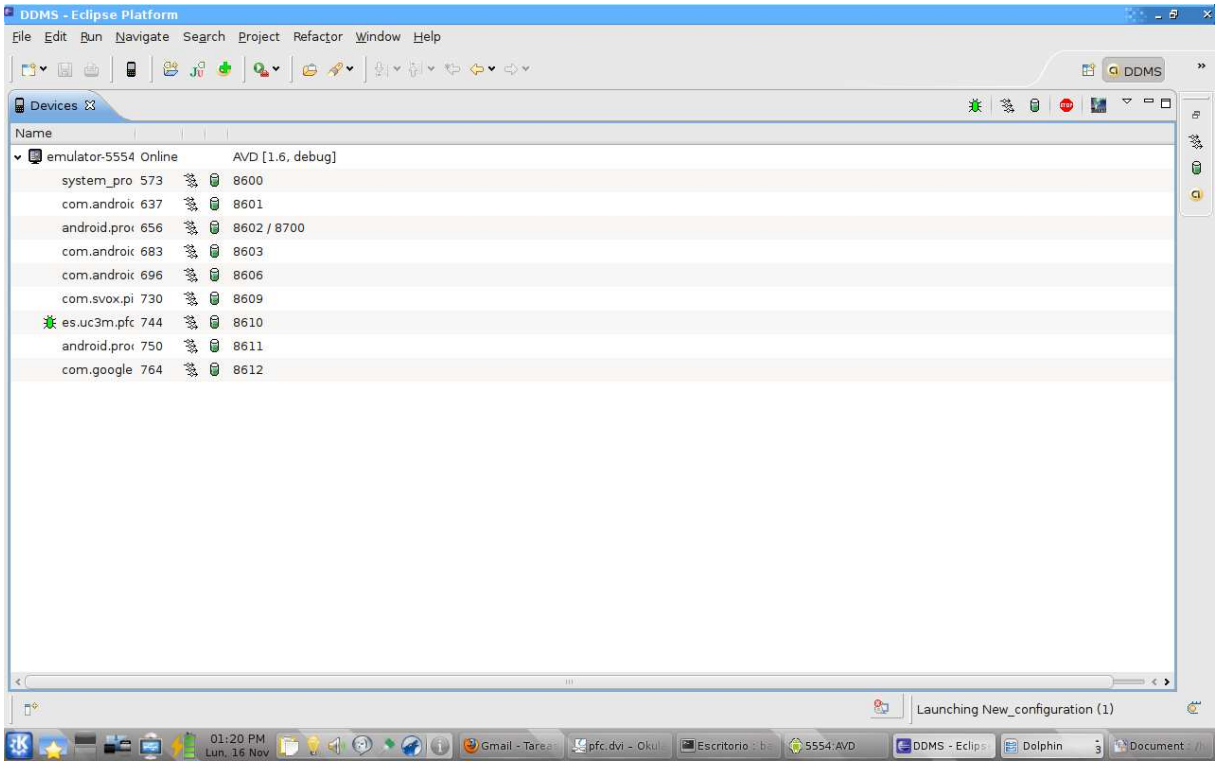

Figura 3.4: Devices

- TID
  Es el identificador de cada una de las hebras. En la hebra principal, TID equivale a ID.

- Estado
  Es el estado de cada una de las hebras. Los posibles estados son:

  1. *running*
     Se está ejecutando código de la aplicación.
  2. *sleeping*
     Se inicia después de la llamada a **Thread.sleep()**.
  3. *monitor*
     La hebra está esperando por el bloqueo del monitor.
  4. *wait*
     Se inicia después de la llamada a **Object.wait()**.
  5. *native*
     Se está ejecutando código nativo.
  6. *vmwait*
     La hebra está esperando por un recurso de la VM.

7. *zombie*
   La hebra está en el proceso de *dying*.
8. *init*
   La hebra se está inicializando.
9. *starting*
   La hebra se va a inicializar.

- Utime
  Es el tiempo total dedicado a ejecutar código de usuario en unidades de tiempo (generalmente 10 ms). Sólo disponible en Linux.
- Stime
  Es el tiempo total dedicado a ejecutar código del sistema en unidades de tiempo (generalmente 10 ms).
- Nombre
  Es el nombre de la hebra.

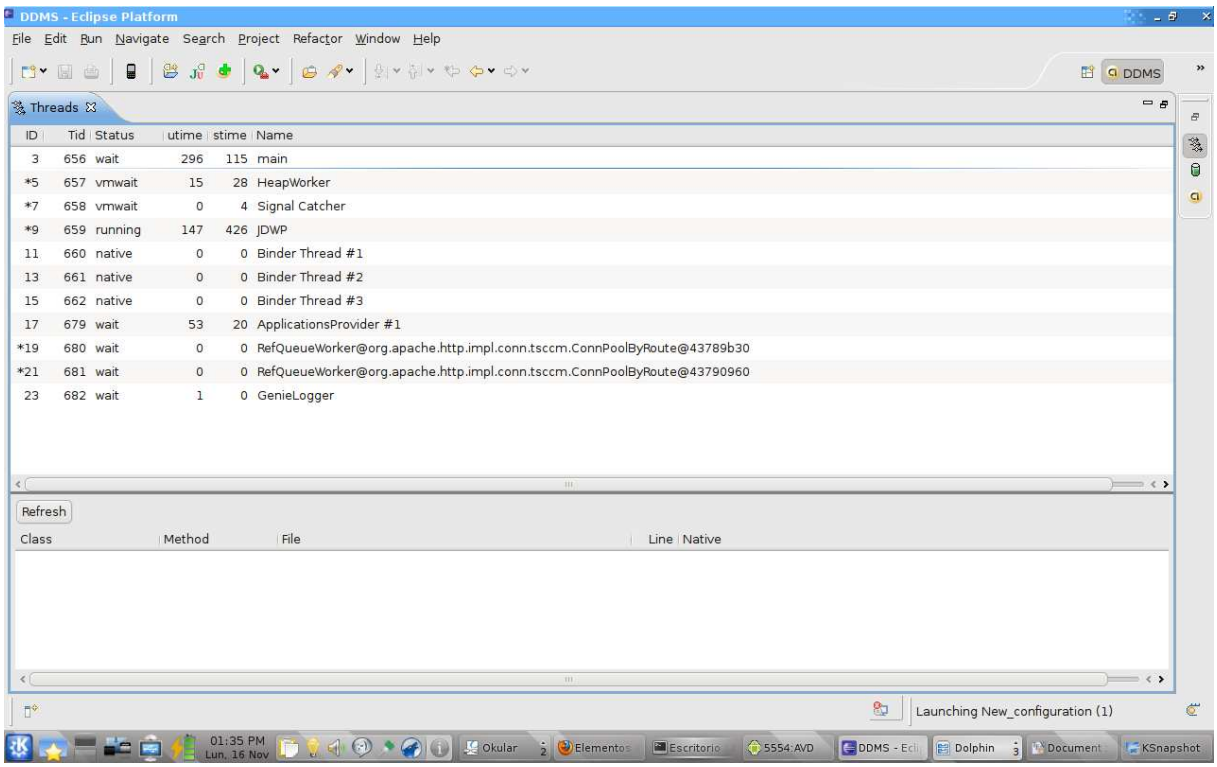

Figura 3.5: Threads

El emulador incluye una variedad de capacidades de depuración que permiten simular interrupciones (enviar mensajes SMS o llamadas telefónicas), modificar datos de localización del

emulador o dispositivo realmente conectado, añadir efectos de latencia, etc. Una vez la aplicación está ejecutándose en el emulador se pueden utilizar cualquiera de los servicios de la plataforma Android: acceso a la red, notificaciones al usuario, etc.

Los apartados principales son:

- Estado telefonía
  Permite simular diferentes tipos de velocidad y latencia de la red (GPRS, EDGE, UMTS, etc).

- Acciones telefonía
  Permite realizar llamadas telefónicas y enviar mensajes SMS desde el emulador.

- Controles situación
  Permite realizar operaciones de localización enviando datos de localización al emulador. Existen varias posibilidades para el envío de estos datos:

  - Enviar manualmente las coordenadas (latitud/longitud). Una de las opciones consiste en seguir los siguientes pasos:
    1. Hacer click en **Manual**.
    2. Seleccionar el formato de coordenadas.
    3. Rellenar los campos habilitados.
    4. Hacer click en **Enviar**.

    La otra opción posible es ejecutar, desde línea de comandos, el siguiente bloque:

    ```
    telnet localhost 5554
    >geo fix lng lat
    ```

  - Utilizar un archivo GPX para describir una ruta para la reproducción en el emulador. Los pasos a seguir son:
    1. Hacer click en **GPX**.
    2. Cargar un archivo GPX.
    3. Hacer click en el botón de reproducción. En este punto se puede ajustar la velocidad y controlar la reproducción con los botones **Pausa** y **Saltar**.

  - Utilizar un archivo KML para describir las posiciones para la reproducción de la secuencia en el emulador.
    1. Hacer click en **KML**.
    2. Cargar un archivo KML.
    3. Hacer click en el botón de reproducción.

Capítulo 4

Otras herramientas

La aplicación web desarollada en este PFC se ha desplegado en el servidor web Jakarta Tomcat ya que es una solución sin coste de licencia y de fácil mantenimiento.

4.1. Jakarta Tomcat

Jakarta Tomcat (también conocido como Apache Tomcat) es un contenedor de servlets, no un servidor de aplicaciones como JBOSS. Los servidores de aplicaciones son capaces de gestionar la mayor parte de las funciones de lógica de negocio y de acceso a los datos de una aplicación empresarial. Si bien el término es aplicable a todas las plataformas de software, hoy en día el término servidor de aplicaciones se ha convertido en sinónimo de la plataforma Java EE de Sun Microsystems.

A pesar del nombre (Apache Tomcat) no depende del servidor web Apache para su funcionamiento; Apache es una implementación en C de un servidor web HTTP que es capaz de servir páginas HTML (existe la posibilidad de añadirle módulos para poder servir páginas PHP, cgi, etc). Aunque en sus inicios existió la percepción de que el uso de Jakarta Tomcat de forma autónoma sólo era recomendable para entornos de desarrollo (con requisitos mínimos de velocidad y gestión de transacciones), la realidad a día de hoy es que Jakarta Tomcat se puede usar como servidor web autónomo en entornos con alto nivel de tráfico y alta disponibilidad.

Sus características principales son:

- Está escrito en Java, por tanto se garantiza su funcionamiento en cualquier sistema operativo que disponga de la máquina virtual Java.
- Implementa las especificaciones de los servlets y de JSPs de Sun Microsystems.
  La versión de Jakarta Tomcat utilizada en este PFC (5.5.27) satisface las especificaciones Servlet 2.4 y JSP 2.0.
- Incluye el compilador Jasper, que proporciona soporte para procesar e interpretar ficheros

JSP y así generar los servlets correspondientes.

- No soporta directamente EJB's ni Web Services pero se puede integrar con soluciones de otros proveedores que proporcionan este tipo de servicios.

4.1.1. Configuración

La configuración de Jakarta Tomcat se basa en dos ficheros:

- server.xml
  Es el fichero de configuración global de los componentes del servidor web. Algunos de sus parámetros son:

  1. server: define un servidor web.
  2. logger: define un objeto Logger que tiene un nombre que lo identifica, el path del fichero de log que contiene la salida y el nivel de depuración.
  3. contextInterceptors: definen los interceptores que escuchan los eventos relacionados con el arranque y la parada del servidor web.
  4. requestInterceptors: definen interceptores que escuchan los eventos relacionados con las peticiones de los usuarios.
  5. context: representa el path de la aplicación web.
  6. connector: define un objeto Connector que es el responsable del control de los *threads* en el servidor y de leer/escribir las peticiones/respuestas desde los sockets conectados a los distintos clientes.
  7. contextManager: especifica la configuración de los *context*, *contextInterceptors*, *requestInterceptors* y *connectors*.

- web.xml
  Es el descriptor de despliegue de la aplicación. Algunos de sus parámetros son:

  1. display-name: define el nombre de la aplicación web.
  2. description: define la descripción de la aplicación web.
  3. listener: identifica la clase Java que se invocará durante el arranque y parada de la aplicación.
  4. welcome-file-list: define el punto de entrada a la aplicación.
  5. servlet: permite declarar un servlet.
  6. servlet-mapping: permite mapear un servlet a una URL.
  7. session-config: permite configurar parámetros de la sesión.

Cualquier aplicación web se resume en un conjunto de archivos agrupados con ciertos parámetros de arranque y seguridad; este conjunto de archivos es desplegado en el servidor web Jakarta Tomcat como un WAR (*Web ARchive*).

4.1.2. Archivos WAR

Un archivo WAR (Web Archive) no es más que un fichero comprimido (al igual que un JAR). Su estructura fue definida por Sun Microsystems para garantizar la portabilidad de las aplicaciones y debe ser implementada en cualquier producto **Servlet Engine** (contenedor de servlets):

- El directorio base contiene la parte pública de la aplicación (documentos HTML, JSP, CSS, código Javascript, imágenes, etc). Normalmente estos archivos se separan en varios subdirectorios.
- El directorio /WEB-INF/classes contiene las clases Java que componen la aplicación.
- El directorio /WEB-INF/lib/ contiene los JAR's necesarios para el correcto funcionamiento de la aplicación.
- El archivo /WEB-INF/web.xml es el descriptor de despliegue de la aplicación.
- El directorio META-INF es opcional y suele contener únicamente el archivo MANIFEST.MF, que indica las bibliotecas de las que depende la aplicación. Este directorio se suele generar automáticamente.

En el momento en el que se ejecuta el script de arranque de Jakarta Tomcat éste inspecciona y automáticamente expande cualquier archivo WAR que encuentra bajo el directorio webapps.

4.1.3. Jakarta Tomcat vs Jetty

Una de las alternativas libres a Jakarta Tomcat es Jetty. Ambos implementan la especificación 2.5 de servlets pero entre ellos existen algunas diferencias:

- Jakarta Tomcat tiende a tener un mejor rendimiento cuando hay pocos usuarios; ésto redunda en una menor latencia de cada solicitud, que es más evidente cuando se envían muchas peticiones/respuestas.
- Jetty tiende a tener una mejor escalabilidad cuando hay muchas conexiones con el tiempo de inactividad importante, como ocurre en la mayoría de los sitios web.
- Jetty tiene un mejor rendimiento en lo que respecta al servicio de contenido estático.

El archivo WAR que representa la aplicación web desarrollada en este PFC se ha generado con la herramienta Ant.

4.2. Ant

En el desarrollo de una aplicación existen algunas tareas que se repiten con frecuencia: compilar, generar el war con el proyecto, etc. Para agilizar este proceso existen varias opciones:

- Ejecutar scripts donde se configuran los comandos a ejecutar.
- Utilizar Ant. Esta opción es equivalente a la anterior para los desarroladores Java (aunque nada impide utilizarlo con otros lenguajes.

Las características principales de esta herramienta son:

- Se basa en ficheros XML (*buildFiles*) que deben estar bien formados, aunque no existe ningún DTD para validarlos. Por ello, para utilizar Ant es necesario disponer de un analizador JAXP (la distribución binaria de Ant incluye la versión más reciente de Xerces).
- Se puede extender utilizando clases Java, lo que le da la capacidad de ser multiplataforma. Es necesario disponer de un JDK instalado en el sistema, versión 1.2 o posterior.
  Cada tarea está asociada a un objeto que implementa la interfaz correspondiente.
- Es fácilmente integrable con otras herramientas. Entornos de desarrollo como Eclipse ofrecen la posibilidad de integrar Ant para que las tareas estén accesibles directamente.

4.2.1. Configuración

El *buildfile* (por defecto Ant usa build.xml como nombre del *buildFile*) debe colocarse en la raiz del proyecto en el que se está trabajando. Algunos de los tags permitidos en este fichero son:

- project: sólo puede haber uno en el fichero, el que se corresponde con la aplicación. Los posibles atributos que puede contener este tag son:
  1. name: nombre del proyecto.
  2. default: *target* que se ejecuta si no se especifica ningún otro.
  3. baseDir: directorio base que se utiliza para definir otras rutas relativas.
- target: conjunto de tareas a ejecutar. Se puede tener un *target* para compilar, otro para crear un fichero war, etc. Los posibles atributos que puede contener este tag son:

1. name: nombre del *target*.
2. if: condición para que se ejecute el *target*.
3. unless: condición para que no se ejecute el *target*.
4. depends: lista de *targets* de los que depende el actual.
5. description: breve descripción del *target*.

- task: código que se puede ejecutar (con N argumentos). Ant incluye muchos *tasks* básicos (**javac**, etc) y permite ejecutar comandos de shell con el *task* **exec**.
- property: pueden ser establecidas en el emphbuildfile o por línea de comandos. Las propiedades, como par nombre-valor, pueden ser utilizadas en el valor de los atributos de un *task*. Algunas de las *properties* que incluye Ant son:

1. ant.file: path absoluto del *buildfile*.
2. ant.java.version: versión de la JVM.
3. ant.version: versión de Ant.

Ant soporta un mecanismo de plug-in para el uso de *tasks* de terceros. Para poder utilizarlos deben ser declarados y su implementación debe ser ubicada en algún punto en el que Ant pueda encontrarla. Existen varias opciones para la declaración:

- declarar una sola tarea:

```
< taskdef name="taskname" classname="ImplementationClass" />
```

- declarar un conjunto de tareas empleando un archivo de propiedades para gestionar los pares *taskname* - *implementationClass*:

```
< taskdef resource="net/sf/antcontrib/antcontrib.properties" />
```

- declarar un conjunto de tareas empleando un archivo XML para gestionar los pares *taskname* - *implementationClass*:

```
< taskdef resource="net/sf/antcontrib/antlib.xml" />
```

- declarar un conjunto de tareas empleando un archivo XML llamado *antlib.xml*, un espacio de nombres XML y la herramienta *antlib*:

```
< project xmlns:ac="antlib:net.sf.antconrib" />
```

Para el desarrollo de aplicaciones Flex se tienen las siguientes herramientas:

- mxmlc: invoca el compilador de la aplicación Flex.
- compc: invoca el compilador de componentes. Se utiliza para compilar los archivos SWC.
- html-wrapper: genera el wrapper HTML para la aplicación Flex y en este punto se pueden especificar algunos de los parámetros de aplicación (altura, anchura, etc).

Aunque dichas herramientas estén en el PATH del sistema, para Ant no son "visibles". Por tanto, para utilizarlas se recurre a la segunda opción de las mencionadas anteriormente: añadir un *task* en el *buildfile* y especificar la localización del archivo flexTasks.jar del SDK de Flex.

Los dos módulos de la solución propuesta en este PFC (aplicación web y aplicación Android) comparten una base de datos relacional, concretamente MySQL, para obtener información de los puntos de acceso WiFi disponibles.

4.3. Bases de datos relacionales

Antes de las bases de datos se utilizaban archivos secuenciales para almacenar los datos. Éstos daban un acceso muy rápido pero sólo de forma secuencial por lo que pronto aparecieron los archivos indexados, donde el acceso ya podía ser aleatorio. Para compartir los datos entre varias máquinas surgió NFS y más tarde, para evitar fallos en los sistemas de archivo, aparecieron los sistemas RAID3.

Poco a poco se iba haciendo necesario un almacenamiento que garantizara algunas condiciones y que permitiera operaciones complejas sin que se violaran estas restricciones. Respondiendo a estas necesidades surgieron las bases de datos jerárquicas en las que el acceso a datos era unidireccional, lo que complicaba el camino inverso. Para dar absoluta libertad a las relaciones entre tablas surgieron las bases de datos relacionales (RDBMS).

Las RDBMS no colocan todos los datos en un gran archivo sino que los mantienen en tablas separadas. Ésto redunda en una alta flexibilidad y en un acceso a los datos más lento. Por otra parte, las bases de datos relacionales trajeron las propiedades ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad):

- Atomicidad
  Cada transacción del usuario debe tratarse de forma atómica, es decir, el trabajo se realiza en su totalidad o no se realiza en ningún caso.
- Consistencia
  Las transacciones han de cumplir las restricciones definidas en la base de datos. Si no las pueden cumplir se cancela su ejecución; de esta forma se conserva la integridad y coherencia de los datos.
- Aislamiento
  Una transacción es una unidad de aislamiento. Si existen N transacciones concurrentes en el sistema, cada una de ellas se comporta como si fuera la única transacción en ejecución. Las transacciones alcanzan el nivel más alto de aislamiento cuando se pueden serializar. En este nivel los resultados obtenidos de un conjunto de transacciones concurrentes son idénticos a los obtenidos mediante la ejecución de las transacciones en serie.
- Durabilidad
  Una vez completada la transacción los resultados de la misma han de ser permanentes y sobrevivir a posibles caídas del sistema o la base de datos.

4.3.1. MySQL

MySQL es un sistema de administración de bases de datos relacional. Inicialmente carecía de elementos considerados esenciales en las RBDMS aunque poco a poco se han ido incorporando. En los últimos años ha tenido un crecimiento vertiginoso, ha pasado de ser una pequeña base de datos a ser la base de datos de código abierto más popular del mundo. A día de hoy compite con sistemas RDBMS propietarios como Oracle, Sql Server y Db2.

MySQL es fácil de instalar, configurar y administrar en la inmensa mayoría de sistemas operativos, por lo que, junto a lenguajes de programación de alta portabilidad como Java, permite el desarrollo de aplicaciones web fáciles de migrar y la transferencia de datos desde cualquier sistema operativo. Su arquitectura se muestra en la figura 4.1.

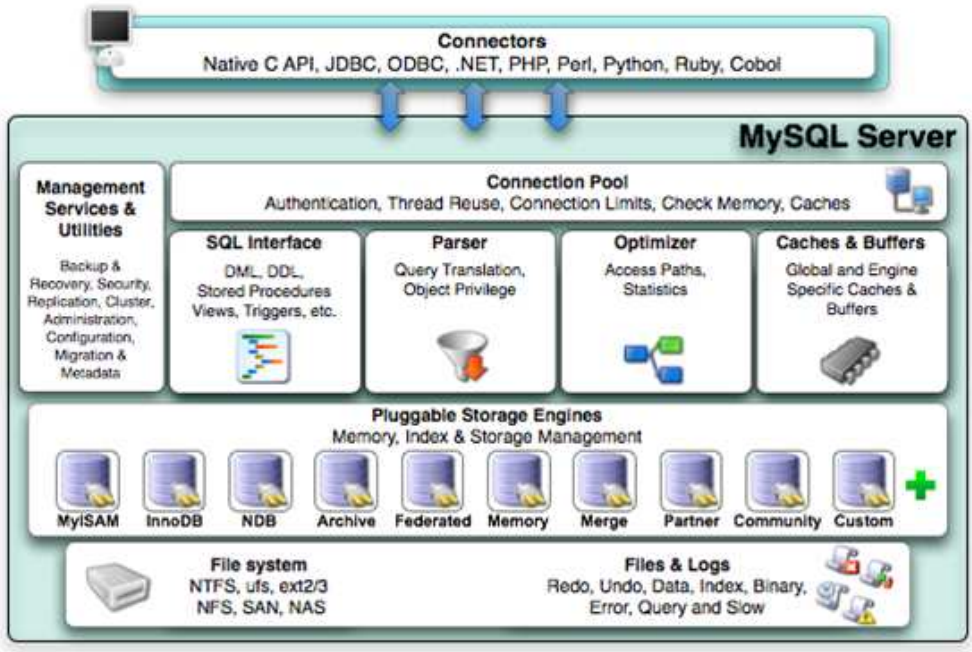

Figura 4.1: Arquitectura MySQL

Algunas de las razones para escoger a MySQL como sistema de administració de bases de datos son:

- Es gratuito para la mayor parte de los usos.
- Está escrito en C y en C++ y probado con múltiples compiladores distintos.
- Ofrece múltiples conectores para el desarrollo de aplicaciones. Cuando el lenguaje empleado en el lado de servidor es PHP, Java, .Net o Perl, MySQL dispone de un driver que implementa la funcionalidad de acceso a datos.

- Es multiplataforma.
- Es mucho más rápido que la mayoría de sus rivales.
- Es multiproceso, puede usar varias CPU si están disponibles.
- Dispone de un motor de almacenamiento InnoDb, que permite trabajar con transacciones y definir reglas de integridad referencial.
- Dispone de un sistema de contraseñas y privilegios muy seguro y flexible que permite verificación basada en el host.
- Se permiten hasta 64 índices por tabla. Cada uno de ellos puede estar compuesto por hasta 16 columnas.
- Los clientes se pueden conectar con el servidor MySQL usando sockets TCP/IP en cualquier plataforma.
  En sistemas WindowsNT, Windows2000, WindowsXP o Windows2003 los clientes pueden usar *named pipes* para la conexión. Por su parte, los clientes de sistemas Unix pueden conectarse a través de sockets Unix.
- Todos los comandos soportan la opción **-help** o **-?** para obtener asistencia en línea.
- El servidor puede proporcionar mensajes de error a los clientes en múltiples idiomas.
- Incluye un amplio subconjunto de instrucciones del lenguaje SQL (*Structured Query Language*). Algunas extensiones son incluidas igualmente.
- Las columnas de las tablas de la base de datos pueden tener valores por defecto.
- Los datos se guardan según el estándar de codificación ISO-8859-1.
- Las columnas de las tablas de la base de datos pueden tener distintos tipos. Algunos de ellos son:

  1. Tipo entero de 1, 2, 3, 4, y 8 bytes.
  2. Tipo coma flotante.
  3. Tipo coma flotante doble precisión.
  4. Tipo carácter.
  5. Tipo fecha.
  6. Tipo enumerado.

- Dispone de *store procedures*, *triggers* y vistas.
- Permite replicación.

MySQL vs Oracle

Oracle es un sistema de administración de base de datos vendido a nivel mundial, aunque la gran potencia que tiene y su elevado precio hace que sólo se vea, por norma general, en empresas multinacionales. Aunque su dominio en el mercado de servidores empresariales ha sido casi total hasta hace poco, recientemente sufre la competencia de gestores de bases de datos comerciales y de la oferta de otros con licencia Software Libre como MySQL.

Para desarrollar en Oracle se utiliza PL/SQL, un lenguaje de programación propio de Oracle de 5º generación, bastante potente para tratar y gestionar la base de datos. Aunque Oracle maneja PL/SQL, la base de datos incluye el compilador Java (que crea código binario) y la JVM (que convierte el código binario en lenguaje máquina) lo que permite escribir procedimientos almacenados, *triggers*, funciones en Java. Los desarrolladores pueden compilar los programas Java directamente en la base de datos o leer una clase java utilizando la utilidad de Oracle llamada **LoadJava**. Por su parte, MySQL no permite ejecutar código Java en la base de datos.

MySQL vs PostgreSQL

PostgreSQL es menos estable que MySQL (con pérdidas de conexión de forma aleatoria). A partir de la versión 8.x se ha observado una notable mejoría en este aspecto aunque en términos de velocidad y rendimiento, MySQL es más recomendable que PostgreSQL.

En algunas ocasiones es necesario acceder a datos de varias tablas de distintas bases de datos. Esto es posible con MySQL pero no con PostgreSQL.

Por todo lo anterior MySQL tiene una cartera de usuarios mucho mayor que PostgreSQL y su código ha sido utilizado ampliamente en entornos de producción.

Para conectar el lenguaje de programación (Java) y la base de datos se utiliza JDBC.

4.4. Introducción a JDBC

JDBC (*Java Database Connectivity*) es un API para la conectividad entre el lenguaje de programación y una amplia gama de bases de datos. El API JDBC está incluido tanto en la plataforma *Java 2 Standard Edition* (J2SE) y la *Java 2 Platform Enterprise Edition* (J2EE) y es independiente del software de administración de base de datos utilizado.

La necesidad de JDBC, a pesar de la existencia de ODBC (*Open Database Connectivity*), viene dada porque ODBC es un interfaz escrito en lenguaje C, que no es un lenguaje portable. Además, el código binario ODBC debe estar cargado en la máquina de cada cliente, al contrario que los drivers basados en la tecnología JDBC. Los drivers JDBC están escritos en Java y por tanto son automáticamente instalables, portables y seguros.

4.4.1. Características principales

JDBC incluye cuatro componentes:

- API JDBC
  El API JDBC se presenta como una colección de interfaces Java y métodos que gestionan la conexión a cada base de datos. Con el API JDBC una aplicación puede ejecutar sentencias SQL haciendo que los cambios se propaguen a la fuente de datos subyacente.
  El API JDBC puede interactuar también con múltiples fuentes de datos en un entorno distribuido.

- Administrador de controladores JDBC
  La sencilla clase **DriverManager** ha sido tradicionalmente la columna vertebral de la arquitectura de JDBC. Una alternativa para establecer la conexión con una fuente de datos es utilizar el objeto **DataSource** registrado con un servicio de nombres JNDI (*Java Naming and Directory Interface*). Este último mecanismo de conexión es el más recomendable porque hace el código más portable y fácil de mantener.

- *JDBC Test Suite*
  Ayuda a probar cualquier controlador JDBC para su conformidad con los requisitos de la *Java 2 Platform Enterprise Edition.*

- *JDBC-ODBC Bridge*
  Proporciona acceso JDBC a través de controladores ODBC. Aún así es recomendable utilizar controladores JDBC para eliminar completamente la configuración de cliente re-

querida por ODBC y evitar que la JVM pueda ser dañada por un error en el código nativo del *bridge*.

Las principales ventajas que ofrece la tecnología JDBC son:

- Aprovechamiento de los datos existentes
  Con la tecnología JDBC las empresas pueden seguir utilizando sus bases de datos (no tienen por qué ser de una arquitectura propietaria) y accediendo a la información fácilmente.
- Desarrollo simplificado
  El API JDBC simplifica considerablemente el desarrollo de aplicaciones porque oculta la complejidad de muchas tareas de acceso a datos y es fácil de implementar. También proporciona acceso a las características subyacentes de la conexión a la base de datos.
- No requiere configuración en el lado del cliente
  Con el API JDBC toda la información necesaria para establecer la conexión con la base de datos viene dada por la URL JDBC. Así se centraliza el mantenimiento del software y se evita la configuración en el lado del cliente.

4.4.2. Clases

A continuación se describen algunas de las interfaces recogidas en el API JDBC:

- DriverManager: es la clase que permite gestionar los drivers instalados en el sistema.
  Un driver proporciona la comunicación entre el API JDBC y la base de datos.
- Connection: es la clase que representa una conexión con una base de datos.
  Una aplicación puede conectarse a N bases de datos.
- Statement: es la clase que permite ejecutar sentencias SQL sin parámetros.
- PreparedStatement: es la clase que permite ejecutar sentencias SQL con algunos parámetros de entrada.
- ResultSet: es la clase que contiene los registros obtenidos al ejecutar una sentencia **SELECT**.

Capítulo 5

Solución propuesta: WiFi Locator

5.1. Introducción

La solución propuesta en este PFC (**WiFi Locator**) se compone de una aplicación web y una aplicación para dispositivos móviles Android.

La aplicación web permite gestionar los puntos de acceso WiFi en un mapa. Ésto es posible mediante la integración con el API de Google Maps para *Flash* (descrito en el Capítulo 2.3.1). Además se pueden efectuar búsquedas para encontrar el AP más próximo a una dirección postal específica utilizando el sistema de codificación geográfica que proporciona Google.

Se ha implementado un control de acceso para que sólo puedan acceder a la aplicación los usuarios registrados. Inicialmente estos usuarios tienen perfil de sólo lectura y únicamente el administrador puede otorgarles permiso de escritura para que puedan llevar a cabo cualquiera de las operaciones disponibles en la herramienta.

Adicionalmente, para alimentar la base de datos del servicio con puntos de acceso ya existentes, se ha añadido una funcionalidad (sólo permitida para el administrador) para importar los APs de la comunidad WiFi de FON.

Por otro lado, la aplicación Android se integra con la base de datos de la parte web, y en base a esta información, permite encontrar puntos de acceso WiFi de dos formas distintas:

- Búsqueda del AP en base a los siguientes criterios:

    1. Más próximo a la posición actual [1].
    2. Más próximo a una región específica.
    3. Mayor potencia en una distancia máxima.
    4. Mayor velocidad en una distancia máxima.

[1] Se requiere un receptor GPS para determinar la posición actual del usuario

- Notificación cuando el usuario entra en el área de cobertura de alguno de los APs registrados.

5.2. Arquitectura

La arquitectura global del servicio se muestra en la Figura 5.1.

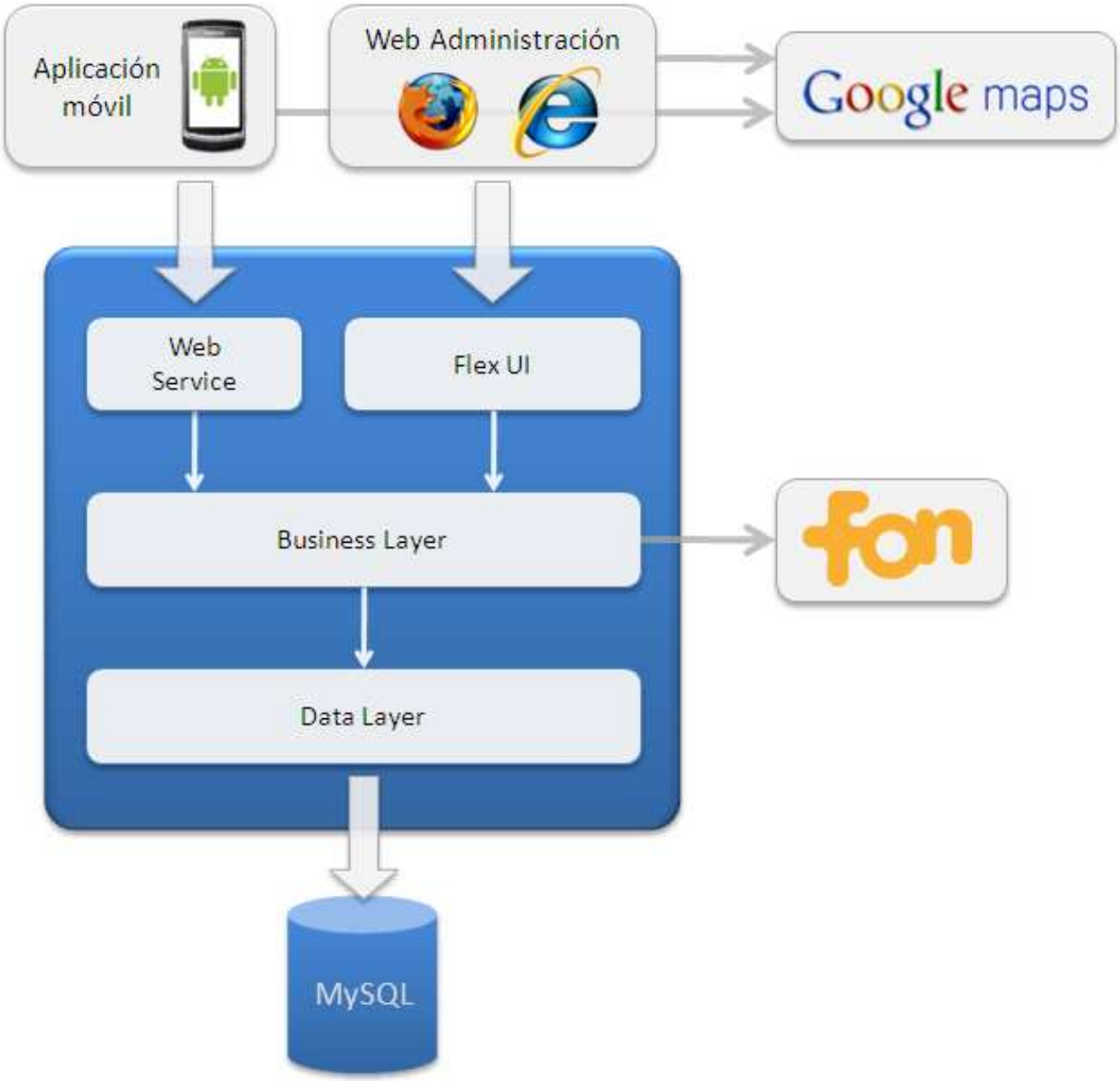

Figura 5.1: Arquitectura del servicio (aplicación Flex y aplicación Android)

- El bloque **Data Layer** se encarga del mapeo de los objetos Java de datos a un entorno persistente (base de datos MySQL).
- El bloque **Business Layer** implementa la lógica de negocio de la solución implementada: gestión de APs, importación de APs de FON, búsqueda por distintos criterios, etc.
- El bloque **Flex UI** representa la interfaz web que se ejecuta en el navegador mediante el plug-in *Flash Player*. La comunicación entre este bloque y *Business Layer* se lleva a cabo mediante el mecanismo descrito en el apartado 2.1.2.
- El bloque **Web Service** representa el servlet a través del que la aplicación Android obtiene la información de los APs. A continuación se muestra el contenido del fichero *web.xml* donde se define la uri de dicho servlet:

```
<?xml version="1.0" encoding="UTF-8"?>
  <web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
                               http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
    <servlet>
      <servlet-name>PfcServerServlet</servlet-name>
      <display-name>PfcServerServlet</display-name>
      <servlet-class>es.uc3m.pfc.server.PfcServerServlet</servlet-class>
      <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
      <servlet-name>PfcServerServlet</servlet-name>
      <url-pattern>/PfcServerServlet/*</url-pattern>
    </servlet-mapping>
</web-app>
```

El contenido de la clase **es.uc3m.pfc.server.PfcServerServlet** se muestra a continuación.

```
package es.uc3m.pfc.server;

import java.io.IOException;
import java.io.OutputStream;
import java.sql.SQLException;
import java.util.Collection;
import java.util.Properties;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import es.uc3m.pfc.PfcService;
import es.uc3m.pfc.model.AP;

public class PfcServerServlet extends HttpServlet {

```

```
17      private static final long serialVersionUID = -5709002957799639884L;
18
19      public void service(HttpServletRequest request, HttpServletResponse response)
20      throws ServletException, IOException {
21
22          OutputStream outputStream = null;
23          // documento XML que enviamos de respuesta a TIBCO
24          response.setContentType("text/txt");
25
26          try {
27              outputStream = response.getOutputStream();
28
29              Properties properties = new Properties();
30              int index = 0;
31
32              Collection<AP> allAPs = new PfcService().getAllAPs();
33              for (AP ap : allAPs) {
34
35                  properties.put(index + ".name", ap.getName());
36                  properties.put(index + ".radius", String.valueOf(ap.getRadius()));
37                  properties.put(index + ".lat", String.valueOf(ap.getLat()));
38                  properties.put(index + ".lng", String.valueOf(ap.getLng()));
39                  properties.put(index + ".txPower", String.valueOf(ap.getTxPower()));
40                  properties.put(index + ".velocity", String.valueOf(ap.getVelocity()));
41                  properties.put(index + ".cyphered", String.valueOf(ap.isCyphered()));
42                  properties.put(index + ".fon", String.valueOf(ap.isFon()));
43
44                  index++;
45              }
46
47              // cargar el properties en el outputStream (el segundo parametro son comentarios)
48              properties.store(outputStream, "");
49
50          } catch (SQLException e) {
51
52          } finally {
53
54          }
55      }
56  }
```

5.3. Diseño de la aplicación web

La url de comunicación, vía AMF, del cliente con el servidor (módulo escrito en Java) se define en el fichero *web.xml*, cuyo contenido se muestra a continuación.

```
1  <filter>
2    <filter-name>AMFMessageFilter</filter-name>
3    <filter-class>org.granite.messaging.webapp.AMFMessageFilter</filter-class>
4  </filter>
5  <filter-mapping>
```

```
6   <filter-name>AMFMessageFilter</filter-name>
7   <url-pattern>/graniteamf/*</url-pattern>
8 </filter-mapping>
9 <servlet>
10  <servlet-name>AMFMessageServlet</servlet-name>
11  <servlet-class>org.granite.messaging.webapp.AMFMessageServlet</servlet-class>
12  <load-on-startup>1</load-on-startup>
13 </servlet>
14 <servlet-mapping>
15  <servlet-name>AMFMessageServlet</servlet-name>
16  <url-pattern>/graniteamf/*</url-pattern>
17 </servlet-mapping>
```

Uno de los ficheros necesarios para establecer la comunicación entre cliente y servidor es *service-config.xml*. Su contenido se muestra a continuación.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <services-config>
3    <services>
4      <service id="granite-service" class="flex.messaging.services.RemotingService"
5               messageTypes="flex.messaging.messages.RemotingMessage">
6        <destination id="PfcService">
7          <channels>
8            <channel ref="my-graniteamf"/>
9          </channels>
10         <properties>
11           <scope>session</scope>
12           <source>es.uc3m.pfc.PfcService</source>
13         </properties>
14       </destination>
15     </service>
16   </services>
17
18   <channels>
19     <channel-definition id="my-graniteamf"
20                         class="mx.messaging.channels.AMFChannel">
21       <endpoint uri="http://localhost:7070/webPfc/graniteamf/amf"
22                 class="flex.messaging.endpoints.AMFEndpoint"/>
23     </channel-definition>
24   </channels>
25 </services-config>
```

Conviene subrayar que los *destination* registrados coinciden con las clases (escritas en Java) expuestas en el lado del servidor. Todos los *destination* emplean el mismo *channel* que define la ruta para la comunicación con el servidor.

Otro de los ficheros necesarios para la configuración de *GraniteDS* es *granite-config.xml*:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE granite-config
3 PUBLIC "-//Granite Data Services//DTD granite-config internal//EN"
```

```
4  "http://www.graniteds.org/public/dtd/1.2.0/granite-config.dtd">
5  <granite-config scan="true">
6    <externalizers>
7      <externalizer type=
8      "org.granite.messaging.amf.io.util.externalizer.DefaultExternalizer">
9        <include type="es.uc3m.pfc.model.AP" />
10     </externalizer>
11   </externalizers>
12 </granite-config>
```

Si se desea mantener cambiar la ubicación de los archivos (*services-config.xml* y *granite-config.xml*) hay que añadir el siguiente bloque en el fichero *web.xml*:

```
1  <context-param>
2    <param-name>servicesConfigPath</param-name>
3    <param-value>/WEB-INF/flex/services-config.xml</param-value>
4  </context-param>
5  <context-param>
6    <param-name>graniteConfigPath</param-name>
7    <param-value>/WEB-INF/granite/granite-config.xml</param-value>
8  </context-param>
```

Una vez establecida la configuración, el cliente Flex debe utilizar un objeto **RemoteObject** para interactuar con el servidor. Desde la clase **PfcService.as** se crea dicho objeto y se invocan sobre él cada una de las operaciones disponibles en el lado del servidor: autenticar, obtener todos los APs de base de datos, añadir/eliminar/modificar un AP, eliminar todos los APS, importar los APs de FON, buscar el AP más próximo a una dirección determinada y mover un AP de un punto a otro.

A continuación se expone la estructura de directorios de la aplicación Flex para entender mejor su funcionamiento.

- **business**
  Las clases contenidas en este paquete componen el módulo que accede al servidor (a través del objeto PfcService mencionado anteriormente).

  En el Apéndice A se muestra (a modo de ejemplo) el contenido de la clase **RegisterUserCommand.as**, que se encarga de la autenticación del usuario que intenta acceder a la aplicación Flex.

- **controller**
  La única clase contenida en este paquete, **PfController.as**, es la encargada de asociar los eventos de la aplicación con los comandos a ejecutar en cada caso, por lo que es fundamental incluirlo en el fichero principal de la aplicación, **pfc.mxml** tal y como se muestra en el Apéndice A.

- **events**
  Las clases contenidas en este paquete se corresponden con los eventos desencadenados por las acciones del usuario sobre el interfaz gráfico.

  En el Apéndice A se presenta (a modo de ejemplo) la clase **RegisterUserEvent.as**, que gestiona el evento que se dispara cuando un usuario accede a la aplicación.

- **model**
  Una de las clases contenidas en este paquete, **Model.as**, contiene información que debe estar accesible desde varias distintas páginas de la aplicación. La otra clase, **AP.as** (su contenido se muestra en el Apéndice A) representa la tabla AP de la base de datos, por tanto sus atributos coinciden con los campos de dicha tabla. Esta clase es generada por el plug-in *GraniteDS* para Eclipse a partir de la clase AP (escrita en Java) del lado del servidor.

- **util**
  En este paquete se encuentran varias clases que optimizan el proceso de visualización de los APs (*markers*) en el mapa (esto es especialmente importante cuando hay un número muy elevado de APs). Existe otra clase, **Util.as**, gestiona las operaciones de añadir, eliminar y mover (*drag and drop*) los *markers* sobre el mapa (ver Apéndice A).

- **view**
  Cada una de las interfaces de usuario de la aplicación (formulario para añadir un AP, formulario para buscar el AP más próximo a una dirección, etc) viene definida en un fichero distinto (todos ubicados en este paquete).

  En el Apéndice A se muestra (a modo de ejemplo) el contenido del fichero **HeaderPanel.mxml** que representa la cabecera de la página principal de la aplicación Flex.

5.4. Interfaz de la aplicación web

Cuando un usuario accede a la url de la aplicación (http://localhost:7070/webPfc/pfc.html) aparece la pantalla que se muestra en la Figura 5.2 para que introduzca sus credenciales (login y password) o se registre en caso de que no lo haya hecho antes. La autenticación de usuarios permite conocer la identidad de la persona que ha realizado la última modificación sobre cada uno de los APs.

Cuando el usuario no especifica el login o la password aparece una ventana emergente que advierte de la obligatoriedad de estos campos (ver Figura 5.3 y Figura 5.4) para que el usuario vuelva a intentarlo si así lo desea. Conviene subrayar que el número de intentos de acceso a la aplicación es ilimitado. Un usuario nunca será bloqueado aunque introduzca incorrectamente sus credenciales en repetidas ocasiones.

Figura 5.2: [Aplicación Flex] Autenticación

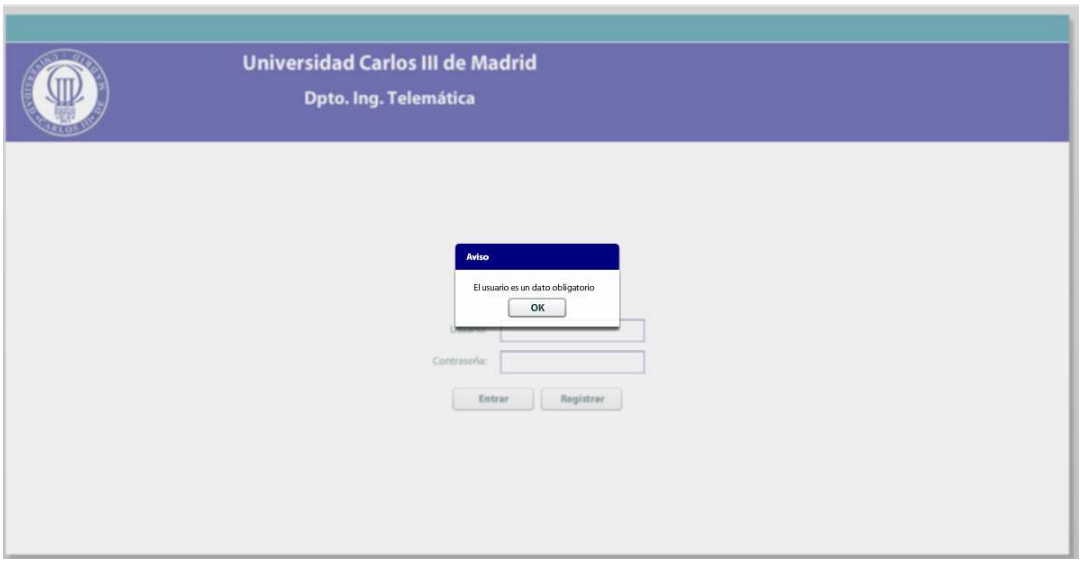

Figura 5.3: [Aplicación Flex] Login no especificado

Cuando el usuario no se autentica correctamente, por haber introducido un login o una password incorrecta, aparece una ventana emergente que lo advierte (ver Figura 5.5) para que el usuario vuelva a intentarlo si así lo desea.

Si la autenticación del usuario es correcta aparece la pantalla principal con todos los APs disponibles en base de datos y la cobertura (se asume un área circular) de cada uno de ellos tal y como se muestra en la Figura 5.6. Para optimizar el proceso de visualización (puede haber un número muy elevado de APs) se irán mostrando según el nivel de zoom del mapa seleccionado por el usuario. Dependiendo de su perfil (lectura/escritura) se habilitarán diversas opciones para

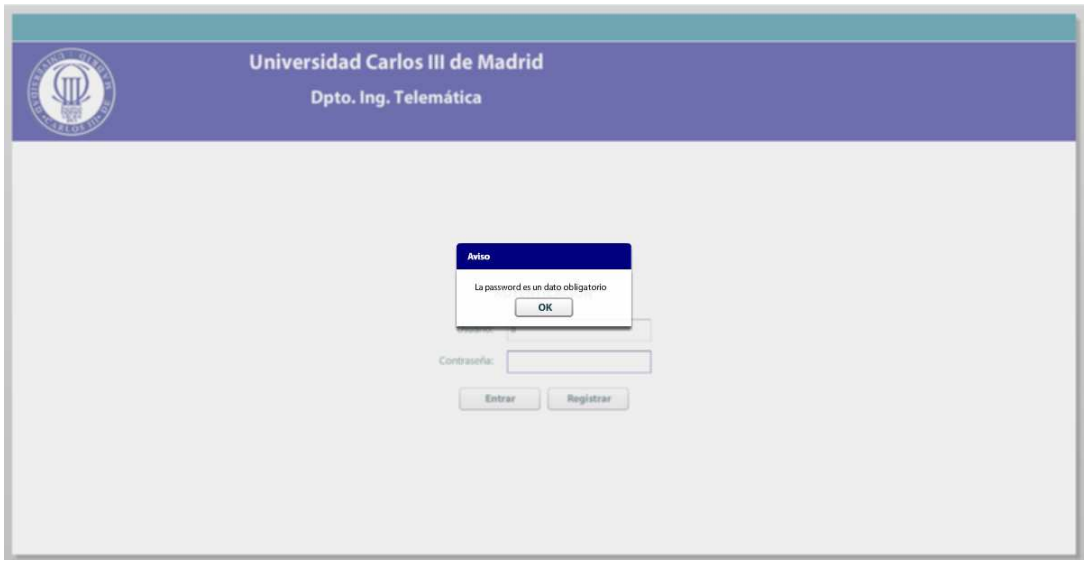

Figura 5.4: [Aplicación Flex] Password no especificada

que el usuario interactúe con el mapa.

Las posibles acciones a realizar son: importar los APs de FON, eliminar todos los APs, añadir un AP en un punto del mapa o en una dirección postal específica, eliminar un AP, modificar las propiedades de un AP, ver los datos de un AP, buscar el AP más próximo a una dirección concreta. Para todas ellos (excepto las dos últimas el usuario debe tener perfil de escritura).

Figura 5.5: [Aplicación Flex] Autenticación fallida

FON es una iniciativa empresarial que surge en el año 2005 con el objetivo de crear una comunidad WiFi global. Esta comunidad permite a sus usuarios la conexión gratuita a los

Figura 5.6: [Aplicación Flex] Pantalla principal

puntos de acceso de otros, repartidos por todo el mundo, a la vez que brinda acceso de pago a terceros mediante un sistema en el que las ganancias se reparten a partes iguales entre la compañía y el usuario que presta su conexión.

Desde la aplicación web se pueden importar todos los APs de FON que aparecerán inmediatamente en el mapa (según el nivel de zom que seleccione el usuario) tal y como se muestra en la Figura 5.7 [2]. A bajo nivel las operaciones realizadas son:

1. Eliminar de la base de datos de la aplicación web todos los APs de FON.
2. Recoger un fichero disponible en el portal de FON.
3. Añadir un AP por cada una de las entradas del fichero anterior.

Si un usuario conoce la localización de algún otro AP, puede añadirlo haciendo doble click en el punto del mapa donde se encuentra el punto de acceso o introduciendo la dirección postal. En ambos casos aparece una ventana emergente (ver Figura 5.8) para completar un formulario con los siguientes datos: nombre (longitud máxima de 100 caracteres), radio de cobertura (en metros), potencia de transmisión (en watios), velocidad (en Mbps) y si es un AP que requiere autenticación o no. En la base de datos (tabla AP) se insertará un nuevo AP con esta información.

Como se ha comentado anteriormente, en la base de datos se almacenará el login del usuario que ha añadido el AP. Por otro lado, el flag que indica si es un AP de FON se almacenará con valor *false* porque el AP ha sido añadido directamente en la herrramienta, no ha sido importado de FON.

[2]La cobertura de los puntos de acceso es representada en el mapa a partir del nivel de zoom 6

Figura 5.7: [Aplicación Flex] Importar APs de FON

Si el valor de alguno de los campos del formulario anterior no es válido, aparece una ventana emergente que advierte del error (ver Figura 5.9) para que el usuario vuelva a intentarlo si así lo desea. No obstante existe la opción *Cancelar* para volver a la página principal.

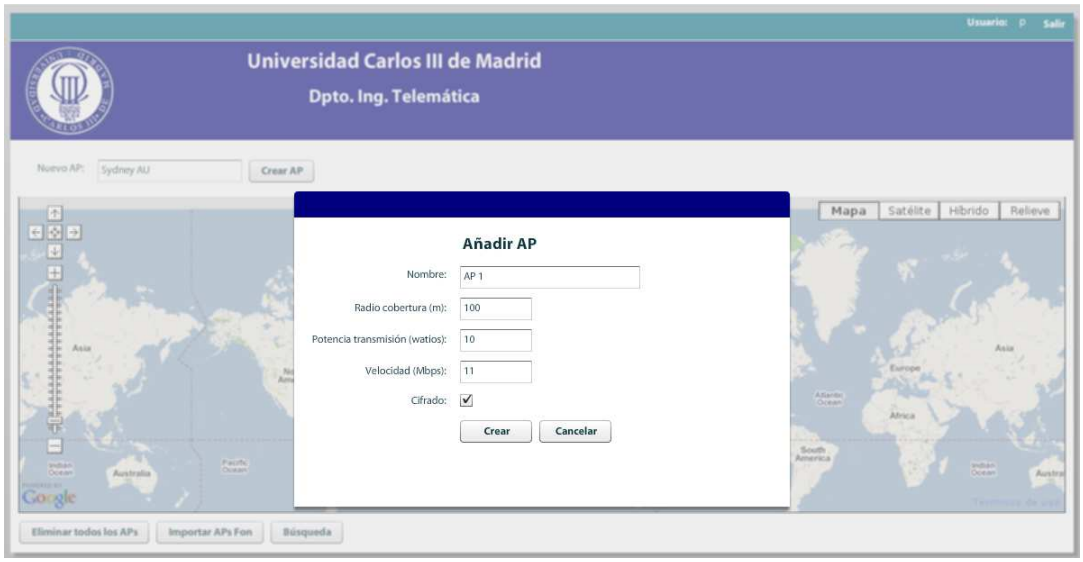

Figura 5.8: [Aplicación Flex] Añadir AP

Al hacer simple click o posicionar el ratón sobre cualquiera de los APs que se muestran en el mapa aparece un *tooltip* con información del AP (nombre, latitud, longitud, radio cobertura, velocidad, etc) tal y como se muestra en la Figura 5.10.

Si un usuario dispone de información acerca de un AP que no se corresponde con lo que hay

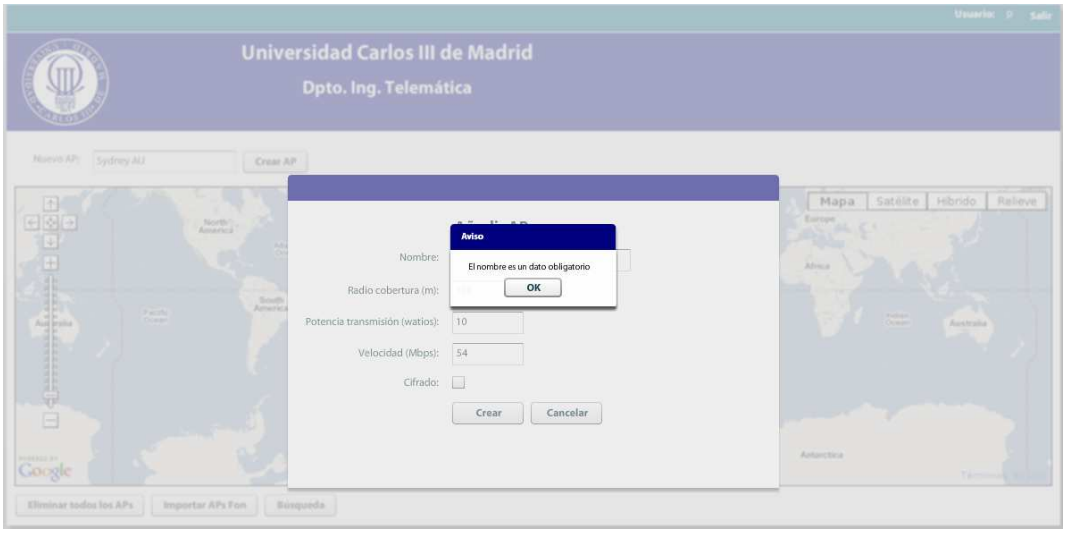

Figura 5.9: [Aplicación Flex] Añadir AP - formulario incorrecto

Figura 5.10: [Aplicación Flex] Ver información AP

en esta herramienta, puede modificar algunas de sus propiedades (radio de cobertura, potencia de transmisión y velocidad). El nombre no se puede editar porque constituye la clave primaria en la base de datos y el flag que indica si es un AP de FON o tampoco.

Si algún AP está emplazado en un punto equivocado, un usuario con el perfil de escritura lo puede eliminar.

Para llevar a cabo cualquiera de las acciones anteriores, modificar/eliminar un AP, el usuario debe hacer doble click en el AP del mapa para que aparezca la ventana emergente que se muestra en la Figura 5.11.

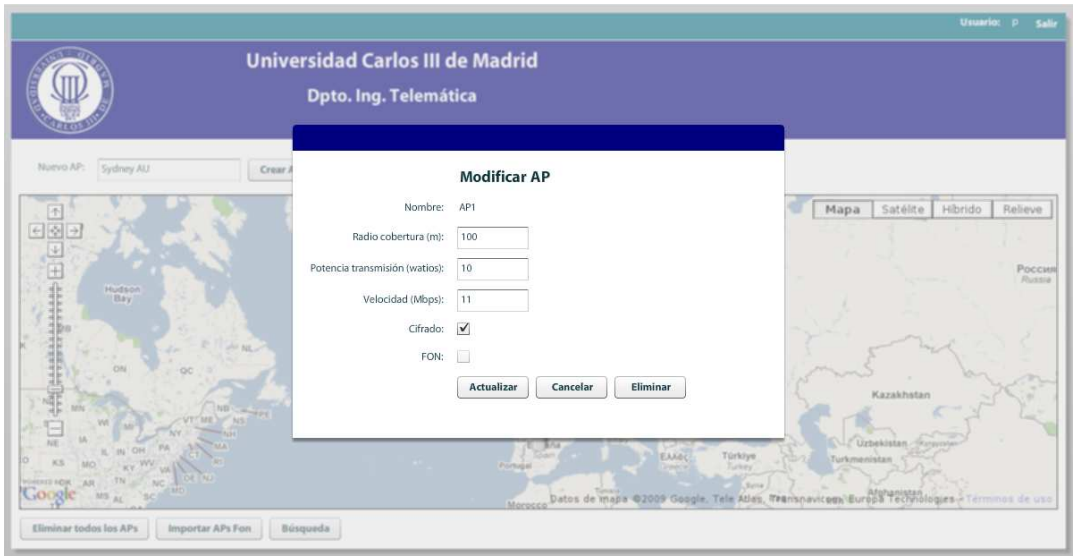

Figura 5.11: [Aplicación Flex] Modificar AP

Si en el formulario anterior se elige la opción *Eliminar* se muestra un diálogo de confirmación (ver Figura 5.12) para que el usuario se detenga un instante antes de continuar. Una vez aceptado, el AP es eliminado de la herramienta de forma irreversible.

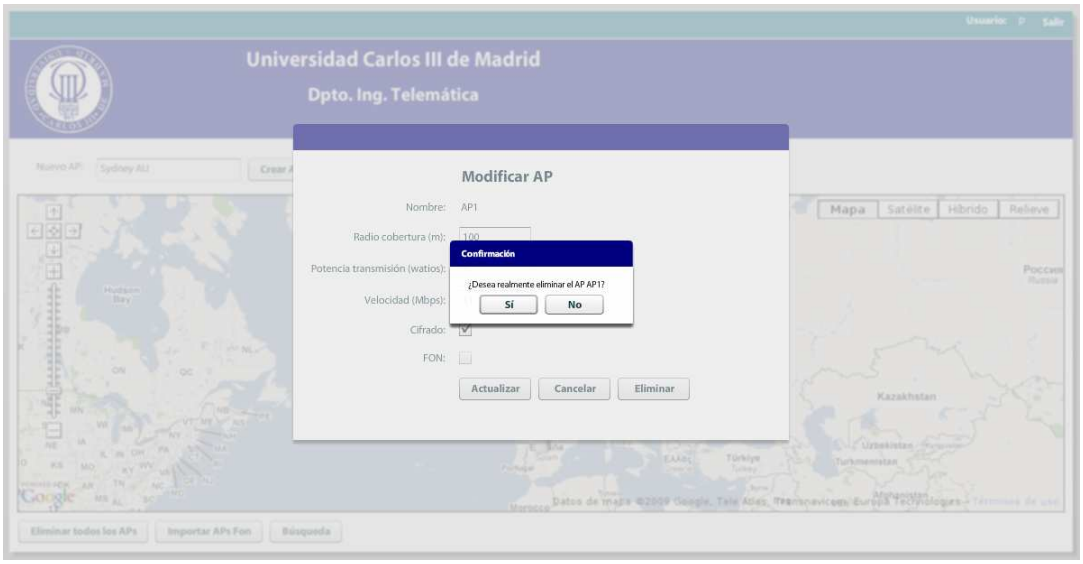

Figura 5.12: [Aplicación Flex] Eliminar AP

Si en un momento dado se quisiera resetear la información de todos los APs disponibles en la herramienta, eliminar cada uno de ellos por separado llevaría bastante tiempo. Para evitarlo, existe la opción *Eliminar todos los APs*. Si se elige esta opción aparece un diálogo de confirmación por el mismo motivo que en el apartado anterior (ver Figura 5.13).

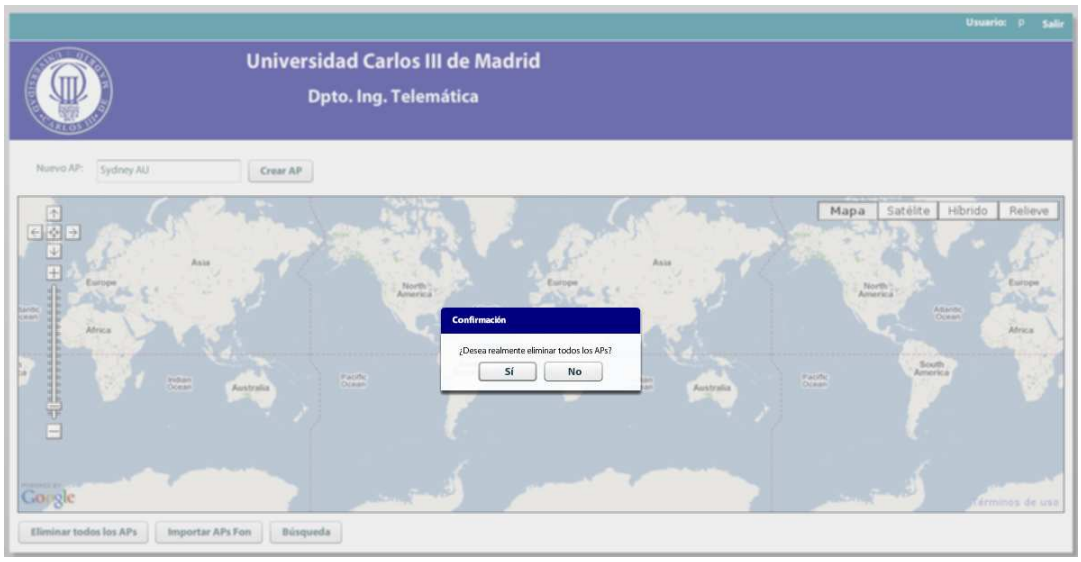

Figura 5.13: [Aplicación Flex] Eliminar todos los APs

Además de interactuar con el mapa el usuario puede consultar la información del AP más próximo a una dirección concreta (ver Figura 5.14). Para ello se recurre a la **codificación geográfica** que traduce una dirección interpretable por humanos en un punto en el mapa. El proceso de conversión o traducción de un punto en una dirección interpretable por humanos se conoce como **codificación geográfica inversa**. Para volver a la página principal existe la opción *Cancelar*.

Figura 5.14: [Aplicación Flex] Interfaz consultas

La **codificación geográfica** no es una ciencia exacta, por tanto, si no se encuentra ninguna

corespondencia para la dirección postal especificada (porque no es válida o porque no existe ningún AP en la base de datos) se informa al usuario de que ha ocurrido un error tal y como se muestra en la Figura 5.15.

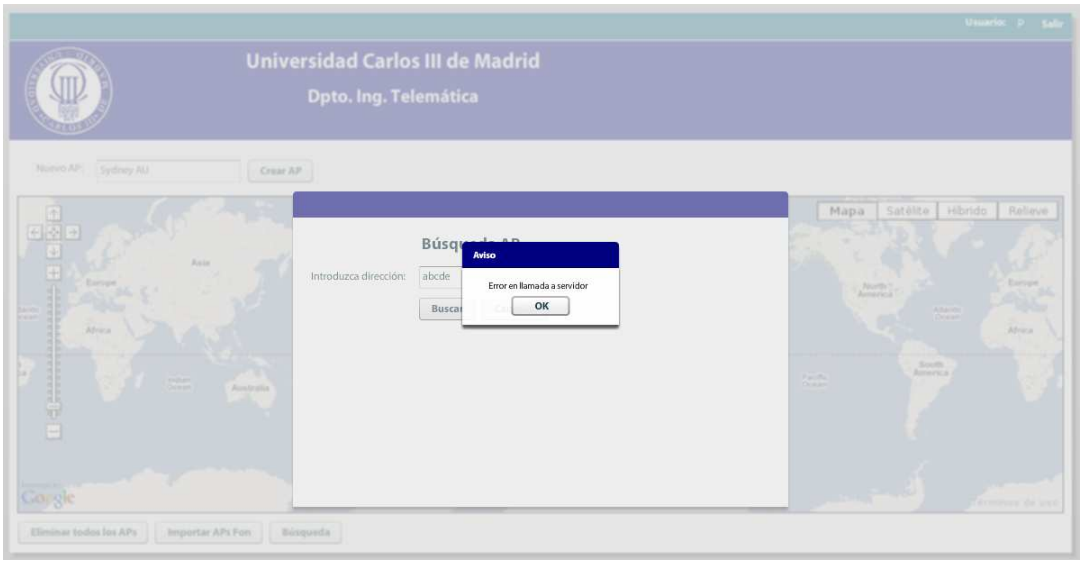

Figura 5.15: [Aplicación Flex] Error en la interfaz consultas

Si la dirección especificada es correcta aparece un formulario con los datos del AP más próximo (ver Figura 5.16). Para volver a la página de consulta existe la opción *Atrás*.

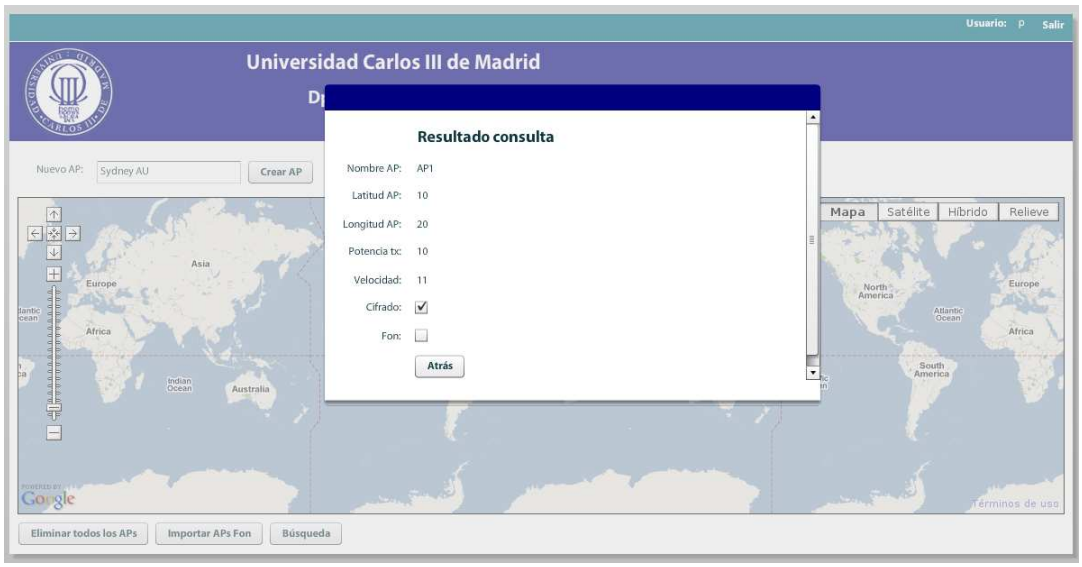

Figura 5.16: [Aplicación Flex] Resultado consulta

Para abandonar la aplicación se puede cerrar directamente la pestaña del navegador, pero para realizar una salida más ordenada existe la opción *Salir* en la parte superior derecha de la

página principal. Si se elige esta opción aparece un diálogo de confirmación para que el usuario se detenga un instante antes de continuar porque una vez aceptado se mostrará la página de login.

Figura 5.17: [Aplicación Flex] Salir de la aplicación

Desde la página de login el usuario tiene la oportunidad de registrarse si no lo ha hecho anteriormente. Si se elige la opción *Registrar* aparece una ventana emergente para completar un formulario con los siguientes datos (ver Figura 5.18): nombre, apellido1, apellido2, dirección, login y password. Todos estos campos son obligatorios y su longitud máxima es 100 caracteres.

Figura 5.18: [Aplicación Flex] Añadir usuario

Si el valor de alguno de los campos del formulario anterior no es válido, aparece una ventana emergente (ver Figura 5.19) que advierte del error para que el usuario vuelva a intentarlo si así lo desea. No obstante existe la opción *Cancelar* para volver a la página de login.

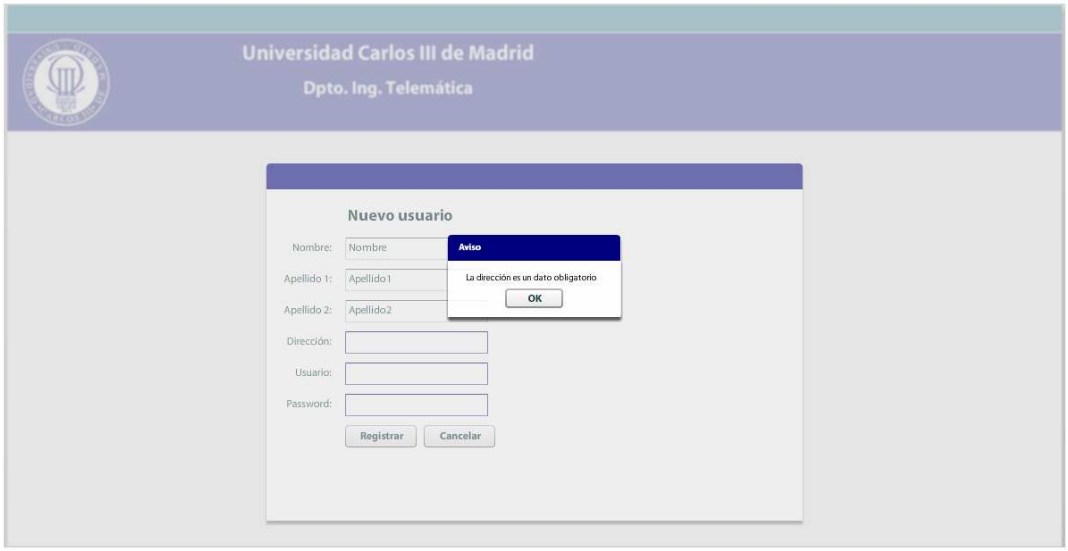

Figura 5.19: [Aplicación Flex] Añadir usuario - formulario incorrecto

Si el usuario rellena el formulario anterior correctamente se inserta una fila en la tabla USER de la base de datos con la información del nuevo usuario. Después de que el usuario se haya registrado correctamente en la herramienta se vuelve a la página de login para que pueda acceder.

5.5. Diseño de la aplicación Android

La aplicación Android tiene en su directorio raiz un archivo llamado *AndroidManifest.xml* cuyo contenido es el siguiente:

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3              package="es.uc3m.pfc"
4              android:versionCode="1"
5              android:versionName="1.0">
6    <application android:icon="@drawable/antena" android:label="@string/app_name">
7      <!-- para usar Google Maps y otras librerias -->
8      <uses-library android:name="com.google.android.maps" />
9      <activity android:name="PfcActivity" android:label="@string/app_name">
10       <intent-filter>
11         <!-- esta es la actividad principal de la aplicacion -->
12         <!-- aparecera en el menu de aplicaciones del sistema -->
13         <action android:name="android.intent.action.MAIN" />
```

```
14          <category android:name="android.intent.category.LAUNCHER" />
15        </intent-filter>
16      </activity>
17      <activity android:name="PfcActivity2"/>
18      <activity android:name="PfcActivity3"/>
19      <activity android:name="PfcMapActivity"/>
20      <activity android:name="PfcBackgroundResultActivity"/>
21    </application>
22    <!-- para usar los servicios GPS -->
23    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
24    <uses-permission android:name="android.permission.INTERNET" />
25    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
26    <uses-sdk android:minSdkVersion="3" />
27  </manifest>
```

Este archivo permite:

- Describir el paquete Java de la aplicación que la identifica de forma unívoca.
- Describir los componentes de la aplicación, indicando el nombre de las clases que los implementan. Así, el sistema Android conoce los componentes que hay y bajo qué condiciones deben ser lanzados.
- Declarar los permisos que debe tener la aplicación para acceder a partes protegidas del API e interactuar con otras aplicaciones.
- Declarar el nivel mínimo del API de Android requiere la aplicación.
- Enumerar las bibliotecas que requiere la aplicación.

Cuando se arranca el emulador de Android se muestran todas las aplicaciones disponibles, entre las que se encuentra la aplicación *Gestión de redes WiFi* (ver Figura 5.20). Cuando se inicia dicha aplicación, se obtienen los APs disponibles en la herramienta ejecutando el siguiente bloque:

```
1  package es.uc3m.pfc;
2
3  import java.io.InputStream;
4  import java.util.Properties;
5  import org.apache.http.HttpEntity;
6  import org.apache.http.HttpResponse;
7  import org.apache.http.client.HttpClient;
8  import org.apache.http.client.methods.HttpGet;
9  import org.apache.http.client.methods.HttpUriRequest;
10 import org.apache.http.impl.client.DefaultHttpClient;
11
12 public class PfcClient {
13
14    public Properties getAPsProperties() {
15
```

```
16        Properties apsProperties = new Properties();
17
18        try {
19           //Interface for an HTTP client.
20           HttpClient httpClient = new DefaultHttpClient();
21           // localhost es el emulador
22           String uri = "http://10.0.2.2:7070/webPfc/PfcServerServlet";
23           HttpUriRequest httpUriRequest = new HttpGet(uri);
24           HttpResponse httpResponse = httpClient.execute(httpUriRequest);
25           // XML de respuesta
26           HttpEntity httpEntity = httpResponse.getEntity();
27           InputStream inputStream = httpEntity.getContent();
28           apsProperties.load(inputStream);
29
30        } catch (Exception e) {
31        }
32        return apsProperties;
33     }
34
35  }
```

Conviene destacar que 10.0.2.2 es un alias especial a la interfaz *loopback*, ya que la dirección IP 127.0.0.1 es la interfaz de *loopback* del emulador Android.

El código que se presenta a continuación se ejecuta por cada uno de los APs que distan menos de 5 Km de la localización actual (para simular un desplazamiento real). Así, cuando el usuario entre dentro del área de cobertura de alguno de ellos recibira una alerta con la información de dicho AP para que se conecte a él si así lo desea.

```
1  Location lastLocation = null;
2  LocationListener myLocationListener = new MyLocationListener();
3  LocationManager locationManager =
4  (LocationManager) getSystemService(Context.LOCATION_SERVICE);
5
6  if (locationManager.isProviderEnabled(LocationManager.GPS_PROVIDER)) {
7     locationManager.requestLocationUpdates(
8                     LocationManager.GPS_PROVIDER, 0, 0, myLocationListener);
9
10    lastLocation = locationManager.getLastKnownLocation(LocationManager.GPS_PROVIDER);
11
12    Intent intentBackgroundResult = new Intent(this, PfcBackgroundResultActivity.class);
13    // If set, this activity will become the start of a new task on this history stack.
14    intentBackgroundResult.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
15
16    // Used in conjunction with FLAG_ACTIVITY_NEW_TASK to disable the behavior of
17    // bringing an existing task to the foreground.
18    intentBackgroundResult.addFlags(Intent.FLAG_ACTIVITY_MULTIPLE_TASK);
19    PendingIntent pendingIntent = PendingIntent.getActivity(PfcActivity.this, index,
20                                                    intentBackgroundResult, 0);
21
22    float[] results = new float[1];
23    Location.distanceBetween(apLat, apLng, lastLocation.getLatitude(),
24    lastLocation.getLongitude(), results);
```

```
25      // maximum distance 5 Km
26      if (results[0] < 5000) {
27          locationManager.addProximityAlert(apLat, apLng, apRadius, 10000000, pendingIntent);
28      }
29  }
```

Figura 5.20: [Aplicación Android] Aplicaciones existentes en el emulador

Conviene subrayar que es necesario implementar la interfaz **LocationListener** para conocer los cambios en la posición GPS del usuario (método **onLocationChanged**). Relacionado con la actualización de la posición del usuario, durante el desarrollo de la aplicación se detectó un bug de Android que se explica detalladamente en el Apéndice B).

```
1   public class MyLocationListener implements LocationListener {
2     public void onLocationChanged(Location loc) {
3       Log.i("PFCActivity", "onLocationChanged: " + loc);
4     }
5
6     public void onProviderDisabled(String provider) {
7       Log.i("PFCActivity", "onProviderDisabled: " + provider);
8     }
9
10    public void onProviderEnabled(String provider) {
11      Log.i("PFCActivity", "onProviderEnabled: " + provider);
12    }
13
14    public void onStatusChanged(String provider, int status, Bundle extras) {
15      Log.i("PFCActivity", "onStatusChanged: provider=" + provider + ", status=" + status);
16    }
17  }
```

En paralelo, aparece en el terminal Android la pantalla que se muestra en la Figura 5.21 para que el usuario pueda seleccionar un criterio (y sólo uno) de búsqueda de AP:

- Por proximidad a la posición actual del usuario.
- Por proximidad a una región específica.
- Por potencia (imponiendo una distancia máxima).
- Por velocidad (imponiendo una distancia máxima).

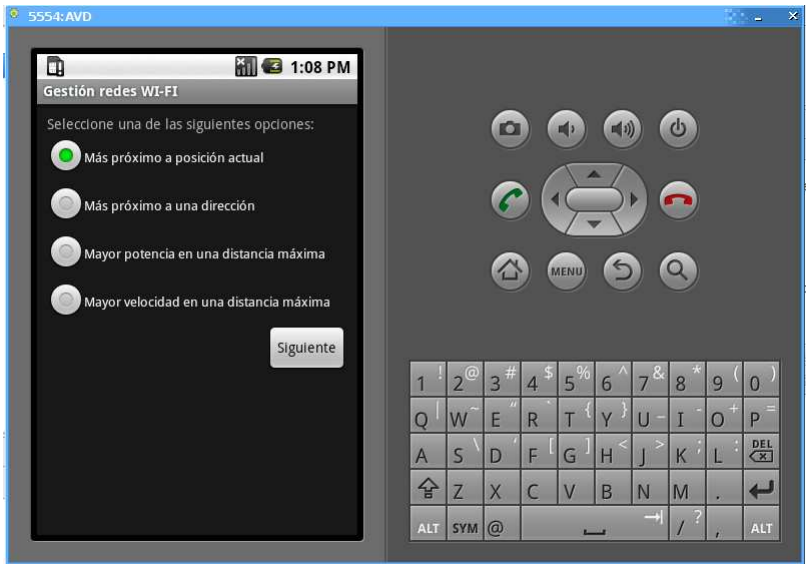

Figura 5.21: [Aplicación Android] Criterios búsqueda AP

Dicha interfaz gráfica está asociada a la clase **PfcActivity**, que extiende la clase **Activity**. Cuando se inicializa la activity se invoca la función **onCreate** donde, entre otras cosas, se especifica el fichero XML que define el *layout*.

```
1 @Override
2 protected void onCreate(Bundle savedInstanceState) {
3   super.onCreate(savedInstanceState);
4   setContentView(R.layout.pfc);
5 }
```

La estructura del fichero layouts/pfc.xml es simple, un árbol de elementos XML donde cada nodo es el nombre de una clase **View** o cualquiera que la extienda. Su contenido se presenta a continuación.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
```

```
3    android:layout_width="fill_parent" android:layout_height="wrap_content"
4    android:padding="10px">
5
6    <TextView android:id="@+id/textView"
7              android:layout_width="fill_parent"
8              android:layout_height="wrap_content"
9              android:text="Seleccione una de las siguientes opciones:" />
10
11   <RadioGroup android:id="@+id/radio_group_inmediate_result_1"
12               android:layout_width="fill_parent"
13               android:layout_below="@+id/textView"
14               android:layout_height="wrap_content"
15               android:orientation="vertical">
16
17   <RadioButton android:id="@+id/radio_button_A_inmediate_result_1"
18                android:layout_width="wrap_content"
19                android:layout_height="wrap_content"
20                android:textSize="13px"
21                android:text="Mas proximo a posicion actual" />
22
23   <RadioButton android:id="@+id/radio_button_B_inmediate_result_1"
24                android:layout_width="wrap_content"
25                android:layout_height="wrap_content"
26                android:textSize="13px"
27                android:text="Mas proximo a una region" />
28
29   <RadioButton android:id="@+id/radio_button_C_inmediate_result_1"
30                android:layout_width="wrap_content"
31                android:layout_height="wrap_content"
32                android:textSize="13px"
33                android:text="Mayor potencia en una distancia maxima" />
34
35   <RadioButton android:id="@+id/radio_button_D_inmediate_result_1"
36                android:layout_width="wrap_content"
37                android:layout_height="wrap_content"
38                android:textSize="13px"
39                android:text="Mayor velocidad en una distancia maxima" />
40   </RadioGroup>
41
42   <Button android:id="@+id/button_next_inmediate_result1"
43           android:layout_width="wrap_content"
44           android:layout_height="wrap_content"
45           android:layout_below="@+id/radio_group_inmediate_result_1"
46           android:layout_alignParentRight="true"
47           android:layout_marginLeft="10px" android:text="Siguiente" />
48   </RelativeLayout>
```

Si en el paso anterior el usuario selecciona la primera opción y pulsa el botón *Siguiente*, aparece una nueva una pantalla (ver Figura 5.22) con las coordenadas de su posición actual. La aplicación Android es capaz de obtener las coordenadas de la posición actual del usuario ejecutando el bloque de código que se expone a continuación [3].

[3]Podría no ser la mas actual si, por ejemplo, el dispositivo esta apagado

```
Location lastLocation = locationManager.getLastKnownLocation(LocationManager.GPS_PROVIDER);
```

Para encontrar los APs más próximos a dicha localización el usuario debe especificar el número de resultados a obtener y pulsar el botón emphBuscar. Como es lógico pensar, el número de resultados debe ser un valor numérico positivo (mínimo 1 y máximo 5). Si desea establecer otro criterio de búsqueda puede volver a la pantalla anterior pulsando el botón *Atrás*.

Figura 5.22: [Aplicación Android] Búsqueda de AP más próximo a la posición actual

Dicha interfaz gráfica está asociada a la clase **PfcActivity2** que, al igual que en el caso anterior, extiende la clase **Activity** y especifica en la función **onCreate** el fichero XML que define el layout.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.pfc2);
}
```

El contenido del fichero layouts/pfc2.xml se presenta a continuación.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="fill_parent" android:layout_height="wrap_content"
        android:padding="10px">
        <TextView android:id="@+id/textView2_0_1"
                android:layout_width="fill_parent" android:layout_height="wrap_content"
                android:text="" />

        <TextView android:id="@+id/textView2_0_2"
```

```
10                  android:layout_width="fill_parent"
11                  android:layout_below="@+id/textView2_0_1"
12                  android:layout_height="wrap_content"
13                  android:text="\nIntroduzca el numero de resultados a obtener" />
14
15          <EditText android:id="@+id/editTextNumber2_0_2"
16                  android:layout_width="40px" android:layout_height="wrap_content"
17                  android:layout_below="@+id/textView2_0_2" android:text="1"
18                  android:lines="1" android:maxLines="1" android:maxLength="1"
19                  android:digits="12345" android:inputType="number" />
20
21          <TextView android:id="@+id/textView2_0_3"
22                  android:layout_width="fill_parent" android:layout_height="wrap_content"
23                  android:text="\nPara buscar el AP mas proximo pulse 'Buscar'"
24                  android:layout_below="@+id/editTextNumber2_0_2" />
25
26          <Button android:id="@+id/buttonSearch2_0" android:layout_width="wrap_content"
27                  android:layout_height="wrap_content"
28                  android:layout_below="@+id/textView2_0_3"
29                  android:layout_alignParentRight="true" android:layout_marginLeft="10px"
30                  android:text="Buscar" />
31
32          <Button android:id="@+id/buttonBack2_0" android:layout_width="wrap_content"
33                  android:layout_height="wrap_content"
34                  android:layout_toLeftOf="@+id/buttonSearch2_0"
35                  android:layout_alignTop="@+id/buttonSearch2_0" android:text="Atras" />
36  </RelativeLayout>
```

Para encontrar los APs más próximos a una localidad concreta, en la nueva pantalla el usuario debe especificar dicha región, el número de resultados a obtener y pulsar el botón *Buscar* (ver Figura 5.23).

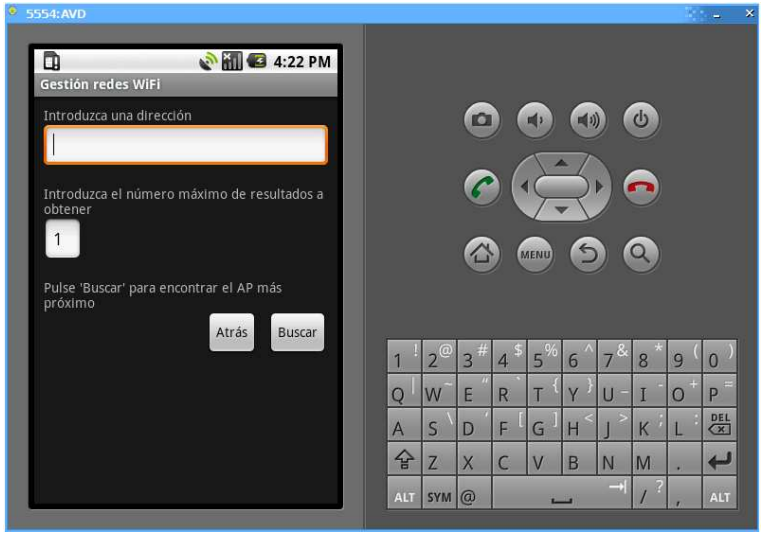

Figura 5.23: [Aplicación Android] Búsqueda de AP más próximo a una región concreta

Geocodificación es el proceso de transformar una dirección (u otra descripción de un lugar) en coordenadas de latitud y longitud. **Geocodificación inversa** es el proceso de transformación de coordenadas en una dirección. La cantidad de detalles en este último caso es variable, puede contener la dirección completa del edificio más cercano o sólo el nombre de la ciudad.

Si la dirección no existe o no existe información de ningún AP en la herramienta, aparece un mensaje de error. En ambos casos, pulsando el botón *Atrás* se vuelve a la pantalla anterior pudiendo establecer otro criterio de búsqueda.

La interfaz gráfica anterior está asociada a la clase **PfcActivity3** que, al igual que en los casos anteriores, extiende la clase **Activity** y especifica en la función **onCreate** el fichero XML que define el layout.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.pfc3);
}
```

El contenido del fichero layouts/pfc3.xml se presenta a continuación.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="fill_parent" android:layout_height="wrap_content"
        android:padding="10px">
        <TextView android:id="@+id/textView2_1_1"
                android:layout_width="fill_parent" android:layout_height="wrap_content"
                android:text="Introduzca una region" />

        <EditText android:id="@+id/editText2_1_1"
                android:layout_width="fill_parent" android:layout_height="wrap_content"
                android:layout_below="@+id/textView2_1_1" android:text=""
                android:lines="1"
                android:maxLines="1" />

        <TextView android:id="@+id/textView2_1_2"
                android:layout_width="fill_parent"
                android:layout_below="@+id/editText2_1_1"
                android:layout_height="wrap_content"
                android:text="\nIntroduzca el numero de resultados a obtener" />

        <EditText android:id="@+id/editTextNumber2_1_2"
                android:layout_width="40px" android:layout_height="wrap_content"
                android:layout_below="@+id/textView2_1_2" android:text="1"
                android:lines="1" android:maxLines="1" android:maxLength="1"
                android:digits="12345" android:inputType="number" />

        <TextView android:id="@+id/textView2_1_3"
                android:layout_width="fill_parent" android:layout_height="wrap_content"
                android:text="\nPulse 'Buscar' para encontrar el AP mas proximo"
                android:layout_below="@+id/editTextNumber2_1_2" />
```

```
31
32          <Button android:id="@+id/buttonSearch2_1" android:layout_width="wrap_content"
33                  android:layout_height="wrap_content"
34                  android:layout_below="@+id/textView2_1_3"
35                  android:layout_alignParentRight="true" android:layout_marginLeft="10px"
36                  android:text="Buscar" />
37
38          <Button android:id="@+id/buttonBack2_1" android:layout_width="wrap_content"
39                  android:layout_height="wrap_content"
40                  android:layout_toLeftOf="@+id/buttonSearch2_1"
41                  android:layout_alignTop="@+id/buttonSearch2_1" android:text="Atras" />
42  </RelativeLayout>
```

Si el usuario desea conectarse a un AP de potencia máxima (en un radio máximo), puede realizar la búsqueda en base al tercer criterio. En la nueva pantalla (ver Figura 5.24) el usuario debe especificar el número de resultados a obtener y la distancia (en metros) en la que se buscará el AP de mayor RSS (*Received Strength Signal*) y pulsar el botón *Buscar*. Como es lógico pensar, la distancia debe ser un valor numérico positivo (mínimo 0 y máximo 9999).

El algoritmo de búsqueda recorre los APs disponibles en la herramienta y por cada uno de ellos:

1. Calcula la distancia entre la posición del AP y la localización actual del usuario (considerando el radio del AP). Si dicha distancia es menor que el límite impuesto va al paso 2 y si no va al paso 3.

2. Compara la potencia de transmisión del AP actual y del AP anterior. Si es un valor superior añade el AP a la lista de resultados y va al paso 3. En caso contrario no selecciona el AP pero también va al paso 3.

3. Va al siguiente AP y vuelve al paso 1. Si no hay mas APs disponibles finaliza el algoritmo.

Si ningún AP satisface la condición de la distancia (paso 1) se seleccionan los APs más próximos a la localización actual del usuario (considerando el radio de cada uno de los APs) hasta alcanzar el número de resultados que haya solicitado el usuario.

Pulsando el botón *Atrás* se vuelve a la pantalla anterior pudiendo establecer otro criterio de búsqueda.

La interfaz gráfica anterior (independientemente del criterio de búsqueda, potencia o velocidad) está asociada a la clase **PfcActivity4** que, al igual que en los otros casos, extiende la clase **Activity** y especifica en la función **onCreate** el fichero XML que define el layout.

```
1  protected void onCreate(Bundle savedInstanceState) {
2    super.onCreate(savedInstanceState);
3    setContentView(R.layout.pfc4);
4  }
```

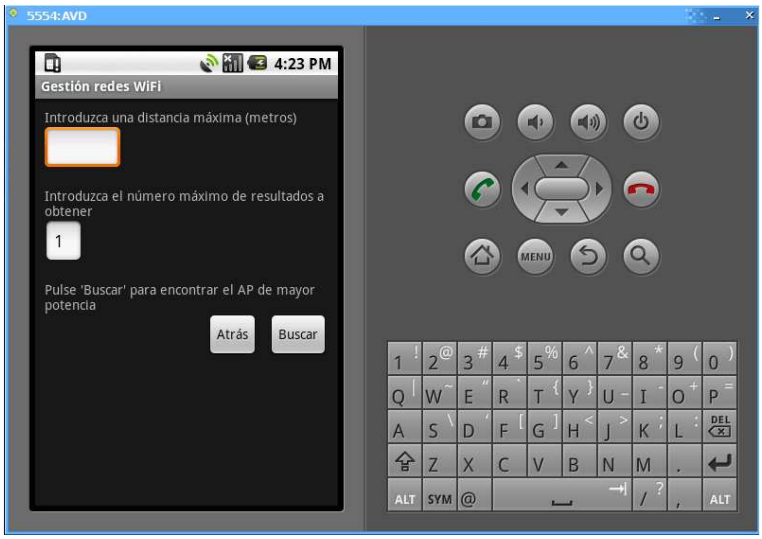

Figura 5.24: [Aplicación Android] Búsqueda de AP de mayor potencia en un radio máximo

El contenido del fichero layouts/pfc4.xml se presenta a continuación.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3          android:layout_width="fill_parent" android:layout_height="wrap_content"
4          android:padding="10px">
5          <TextView android:id="@+id/textView2_2_1"
6                  android:layout_width="fill_parent" android:layout_height="wrap_content"
7                  android:text="Introduzca una distancia maxima (metros)" />
8
9          <EditText android:id="@+id/editTextNumber2_2_1"
10                 android:layout_width="80px" android:layout_height="wrap_content"
11                 android:layout_below="@+id/textView2_2_1" android:text=""
12                 android:lines="1" android:maxLines="1" android:maxLength="4"
13                 android:digits="0123456789." android:inputType="number" />
14
15         <TextView android:id="@+id/textView2_2_2"
16                 android:layout_width="fill_parent"
17                 android:layout_below="@+id/editTextNumber2_2_1"
18                 android:layout_height="wrap_content"
19                 android:text="\nIntroduzca el numero de resultados a obtener" />
20
21         <EditText android:id="@+id/editTextNumber2_2_2"
22                 android:layout_width="40px" android:layout_height="wrap_content"
23                 android:layout_below="@+id/textView2_2_2" android:text="1"
24                 android:lines="1" android:maxLines="1" android:maxLength="1"
25                 android:digits="12345" />
26
27         <TextView android:id="@+id/textView2_2_3"
28                 android:layout_width="fill_parent" android:layout_height="wrap_content"
29                 android:text="\nPulse 'Buscar' para encontrar el AP de mayor
30                         potencia/velocidad"
31                 android:layout_below="@+id/editTextNumber2_2_2" />
```

```
32
33          <Button android:id="@+id/buttonSearch2_2" android:layout_width="wrap_content"
34                  android:layout_height="wrap_content"
35                  android:layout_below="@+id/textView2_2_3"
36                  android:layout_alignParentRight="true" android:layout_marginLeft="10px"
37                  android:text="Buscar" />
38
39          <Button android:id="@+id/buttonBack2_2" android:layout_width="wrap_content"
40                  android:layout_height="wrap_content"
41                  android:layout_toLeftOf="@+id/buttonSearch2_2"
42                  android:layout_alignTop="@+id/buttonSearch2_2" android:text="Atras" />
43  </RelativeLayout>
```

Si el usuario desea conectarse a un AP de velocidad máxima (en un radio máximo), puede realizar la búsqueda en base al cuarto criterio. En la nueva pantalla (ver Figura 5.25) el usuario debe especificar el número de resultados a obtener y la distancia (en metros) en la que se buscará el AP de mayor velocidad y pulsar el botón *Buscar*.

El algoritmo de búsqueda recorre todos los APs y por cada uno de ellos:

1. Calcula la distancia entre la posición del AP y la localización actual del usuario (considerando el radio del AP). Si dicha distancia es menor que el límite impuesto va al paso 2 y si no va al paso 3.

2. Compara la velocidad del AP actual y del AP anterior. Si es un valor superior añade el AP a la lista de resultados y va al paso 3. En caso contrario no selecciona el AP pero también va al paso 3.

3. Va al siguiente AP y vuelve al paso 1. Si no hay mas APs disponibles finaliza el algoritmo.

Igual que en el caso anterior, si ningún AP satisface la condición de la distancia (paso 1) se seleccionan los APs más próximos a la localización actual del usuario (considerando el radio del AP) hasta alcanzar el número de resultados que haya solicitado el usuario.

Sea cual sea el criterio (proximidad a la localización actual, proximidad a una región específica, maxima potencia, maxima velocidad), una vez finalizada la búsqueda aparece una pantalla con la información de los APs que mejor satisfacen las condiciones impuestas (ver Figura 5.26).

Dicha interfaz gráfica está asociada a la clase **PfcActivity5** que, al igual que en los otros casos, extiende la clase **Activity** y especifica en la función **onCreate** el fichero XML que define el layout.

```
1  @Override
2  protected void onCreate(Bundle savedInstanceState) {
3    super.onCreate(savedInstanceState);
4    setContentView(R.layout.pfc5);
5  }
```

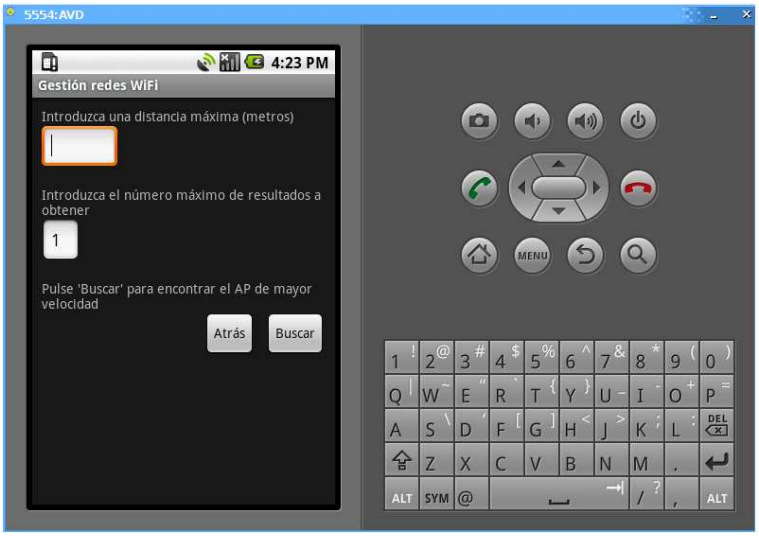

Figura 5.25: [Aplicación Android] Búsqueda de AP de mayor velocidad en un radio máximo

Figura 5.26: [Aplicación Android] Información de los APs que mejor satisfacen el criterio de búsqueda

El contenido del fichero layouts/pfc5.xml se presenta a continuación.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3          android:layout_width="fill_parent" android:layout_height="wrap_content"
4          android:padding="10px">
5          <ScrollView android:id="@+id/ScrollView01"
6                  android:layout_height="300px"
7                  android:layout_width="fill_parent">
8                  <TextView android:id="@+id/textView3"
9                      android:layout_width="fill_parent"
10                     android:layout_height="wrap_content" android:text="" />
```

```
11          </ScrollView>
12          <Button android:id="@+id/buttonViewMap3"
13              android:layout_width="wrap_content"
14              android:layout_height="wrap_content"
15              android:layout_below="@+id/ScrollView01"
16              android:layout_alignParentRight="true"
17              android:layout_marginLeft="10px"
18              android:text="Ver mapa" />
19          <Button android:id="@+id/buttonBack3"
20              android:layout_width="wrap_content"
21              android:layout_height="wrap_content"
22              android:layout_toLeftOf="@+id/buttonViewMap3"
23              android:layout_alignTop="@+id/buttonViewMap3" android:text="Atras" />
24  </RelativeLayout>
```

Por otro lado, cuando el usuario entra en el área de cobertura de cualquier AP aparece en el terminal Android una ventana emergente (con un resultado) por cada uno de ellos (ver Figura 5.27). Esta interfaz gráfica es prácticamente idéntica a la anterior.

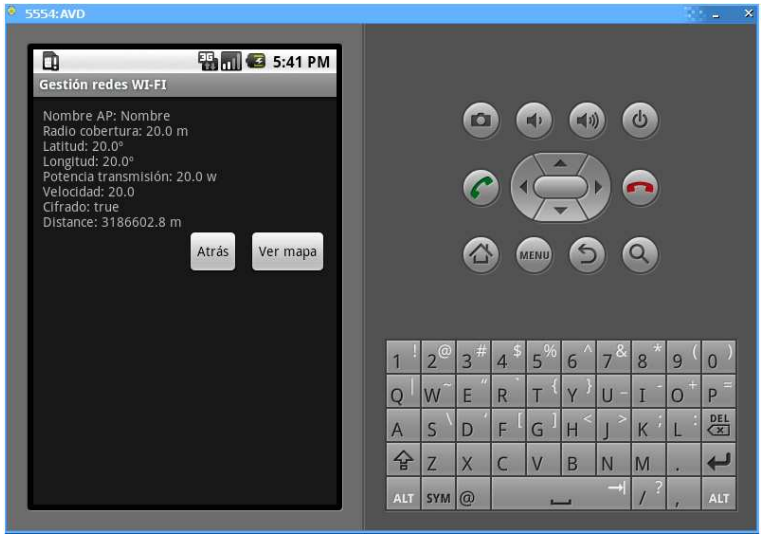

Figura 5.27: [Aplicación Android] Información del AP en cuyo área de cobertura ha entrado el usuario

En cualquiera de los casos, los datos que se proporcionan son: nombre, radio cobertura, potencia transmisión, velocidad, cifrado y distancia a la posición actual del usuario. Si el usuario desea visualizar en un mapa la localización del AP debe pulsar el botón *Ver mapa*. Si por el contrario, el usuario desea repetir la búsqueda (con el mismo u otro criterio) debe pulsar el botón *Atrás*.

Cuando se pulsa el botón *Ver mapa* el usuario puede visualizar en un mapa su posición actual y la ubicación de los APs que mejor satisfacen el criterio de búsqueda seleccionado (ver Figura 5.28).

Figura 5.28: [Aplicación Android] Mapa con la posición actual del usuario y la de los APs seleccionados

La interfaz gráfica anterior está asociada a la clase **PfcActivity6** que extiende la clase **MapActivity**. La función principal se presenta a continuación.

```
1  @Override
2  protected void onCreate(Bundle savedInstanceState) {
3
4      super.onCreate(savedInstanceState);
5
6      requestWindowFeature(Window.FEATURE_NO_TITLE);
7
8      FrameLayout frame = new FrameLayout(this);
9
10     // View which displays a map (with data obtained from the Google Maps service)
11     mapLocationViewer = new MapLocationViewer(this,
12                             "0VsLB5aPhWYjfSMidK7b8shiMEqPwY3E9J8o1AQ");
13
14     // Add map to frame
15     frame.addView(mapLocationViewer);
16
17     setContentView(frame);
18
19     mapController = mapLocationViewer.getController();
20
21     mapLocationViewer.setTraffic(true);
22     mapLocationViewer.setSatellite(false);
23     mapController.setZoom(3);
24
25     View zoomControls = mapLocationViewer.getZoomControls();
26
27     FrameLayout.LayoutParams p = new FrameLayout.LayoutParams(LayoutParams.WRAP_CONTENT,
28             LayoutParams.WRAP_CONTENT,
29             Gravity.BOTTOM + Gravity.CENTER_HORIZONTAL);
```

```
30
31      // Add zoom controls to frame
32      frame.addView(zoomControls, p);
33
34      Intent previousIntent = this.getIntent();
35
36      double lat = previousIntent.getDoubleExtra("es.uc3m.pfc.map.lat", 0);
37      double lng = previousIntent.getDoubleExtra("es.uc3m.pfc.map.lng", 0);
38
39      // Represents a pair of latitude and
40      // longitude (microdegrees) of user current location
41      GeoPoint userPoint = new GeoPoint((int) (lat * 1E6), (int) (lng * 1E6));
42      mapLocationViewer.getManager().addMapLocation(
43         new MapLocation(mapLocationViewer, "User is here",
44                         userPoint, MapLocation.TYPE_USER));
45
46      // Center the map on the user position
47      mapController.setCenter(userPoint);
48
49      // Selected APs
50      int numResults = previousIntent.getIntExtra("es.uc3m.pfc.map.numResults", 0);
51
52      for (int i = 0; i < numResults; i++) {
53         double apLat = previousIntent.getDoubleExtra("es.uc3m.pfc.map.apLat." + i, 0);
54         double apLng = previousIntent.getDoubleExtra("es.uc3m.pfc.map.apLng." + i, 0);
55         boolean apCyphered = previousIntent.getBooleanExtra(
56                              "es.uc3m.pfc.map.apCyphered." + i, false);
57         // AP radius
58         float apRadius = previousIntent.getFloatExtra("es.uc3m.pfc.map.apRadius." + i, 0);
59
60         // AP  name
61         String apName = previousIntent.getStringExtra("es.uc3m.pfc.map.apName." + i);
62
63         // Represents a pair of latitude and
64         // longitude (microdegrees) of AP
65         GeoPoint apPoint = new GeoPoint((int) (apLat * 1E6), (int) (apLng * 1E6));
66
67         // Cyphered AP
68         if (apCyphered) {
69            mapLocationViewer.getManager().addMapLocation(
70               new MapLocation(mapLocationViewer,
71                               "AP: " + apName + ", radius: " + apRadius, apPoint,
72                               MapLocation.TYPE_CYPHERED));
73
74         // No cyphered AP
75         } else {
76            mapLocationViewer.getManager().addMapLocation(
77               new MapLocation(mapLocationViewer,
78                               "AP: " + apName + ", radius: " + apRadius, apPoint,
79                               MapLocation.TYPE_NO_CYPHERED));
80         }
81      }
82   }
```

Sobre el mapa se ha añadido un control para que el usuario pueda hacer zoom (acercar o alejar) sobre el mapa. Pulsando en cada uno de los APs, aparece un *bocadillo* donde se muestra el nombre y el radio del AP (en metros). En el API de Android no existe una forma sencilla de crearlos, por ello se ha utilizado un componente Android desarrollado de forma separada.

Dependiendo de si los puntos de acceso son cifrados o no se muestra un icono distinto. Así, aunque la información de los APs también se presenta en modo texto, el usuario puede diferenciarlos visualmente.

Es de mencionar que durante el desarrollo se intentó representar el área de cobertura de cada uno de los APs sobre el mapa. Los datos disponibles para ello son: *span* de latitud (en º) que se muestra en el mapa, propiedad *height* del mapa (en pixels) y radio del AP (en metros aunque se puede convertir a º).

El *span* de latitud (en º) que se muestra en el mapa se obtiene con una de las funciones disponibles en el API de Android. Este valor no se actualiza hasta la completa renderización del mapa, lo que provoca que el área circular no se llegue a mostrar. Por ello este punto se deja como una posible línea de mejora del presente PFC.

Para salir ordenadamente de la aplicación el usuario debe pulsar en el botón *Salir* que aparece cuando el usuario pulsa en el botón *Menú*.

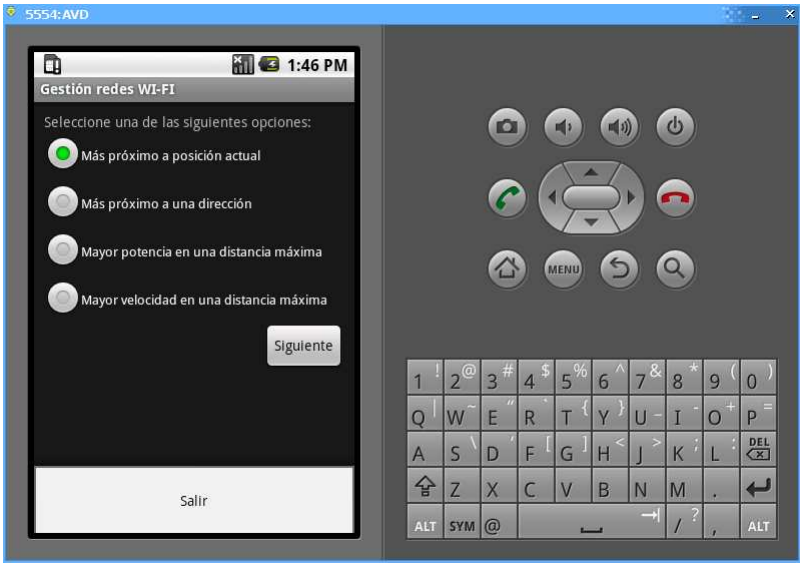

Figura 5.29: [Aplicación Android] Salir ordenadamente de la aplicación

5.6. Modelo de datos

El modelo de datos de la solución propuesta en este PFC es muy simple. Consta de dos tablas, cuyos *scripts* de creación se presentan a continuación.

```
CREATE TABLE `AP` (
  `ID` int(11) NOT NULL auto_increment,
  `NAME` varchar(100) default NULL,
  `RADIUS` float default NULL,
  `LAT` double default NULL,
  `LNG` double default NULL,
  `TX_POWER` double default NULL,
  `VELOCITY` double default NULL,
  `CYPHERED` tinyint(1) default '0',
  `IS_FON` tinyint(1) default '0',
  `LOGIN` varchar(100) default NULL,
  PRIMARY KEY  (`ID`)
) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=latin1
```

- NAME: nombre del AP; constituye la clave primaria, por tanto no se permitirá el mismo nombre para dos APs distintos.
- RADIUS: radio del AP (en metros).
- LAT: latitud del AP (en º).
- LNG: longitud del AP (en º).
- TX_POWER: potencia de transmisión del AP (en watios).
- CYPHERED: flag que indica si el AP implementa algún mecanismo de cifrado para limitar el control de acceso.
- IS_FON: flag que indica si es un AP de FON.
- LOGIN: login del usuario que ha realizado la última modificación sobre un AP determinado.

```
CREATE TABLE `USER` (
  `NAME` varchar(100) default NULL,
  `SURNAME1` varchar(100) default NULL,
  `SURNAME2` varchar(100) default NULL,
```

```
  `ADDRESS` varchar(100) default NULL,
  `LOGIN` varchar(100) NOT NULL default '',
  `PASSWORD` varchar(100) default NULL,
  `ROLE` int(11) default NULL,
  PRIMARY KEY  (`LOGIN`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

- NAME: nombre del usuario.
- SURNAME1: primer apellido del usuario.
- SURNAME2: segundo apellido del usuario.
- ADDRESS: dirección del usuario.
- LOGIN: login que identifica al usuario en la herramienta.
- PASSWORD: password del usuario en la herramienta.

Capítulo 6

Conclusiones

6.1. Comparación con otras soluciones

Las redes inalámbricas se expanden con mucha rapidez, tanto en las casas como en las oficinas y espacios públicos. Existen numerosas aplicaciones que permiten conocer la existencia de estas redes y/o detectarlas cuando cuando se está dentro de su área de cobertura.

A continuación se presentan algunas de estas aplicaciones, analizando sus ventajas e inconvenientes frente a la solución propuesta en este PFC.

6.1.1. Aplicaciones de detección de redes WiFi

Este tipo de aplicaciones se apoyan en un adaptador WiFi que permite detectar las redes a las que el usuario puede acceder desde su localización actual. Algunas de ellas se describen a continuación.

- Gestor WiFi del sistema operativo
  Los principales sistemas operativos incorporan gestores de redes WiFi que permiten detectar y conectarse a las redes WiFi disponibles. Sin embargo resultan bastante básicos ya que sólo permiten conocer (sin demasiada precisión) el nivel de potencia de la señal recibida.

- Soluciones propietarias de las tarjetas WiFi
  Las tarjetas inalámbricas suelen venderse junto a un programa que permite gestionarlas. Estas soluciones suelen proporcionar información más detallada que la anterior (nombre y dirección MAC del router, etc) pero aún así comparte las mismas limitaciones.

- Xirrus WiFi Monitor ([34])
  Esta aplicación se utiliza como un *widget*, una pequeña ventana que se sobreimpresiona en el escritorio. Xirrus ofrece un radar que permite comprobar de un vistazo dónde se sitúan

las mejores redes en función de la potencia de señal con la que se capten. Existen versiones para Windows Vista y GNU/Linux y, a través de Yahoo! Widgets, para Windows XP y Mac Os X.

Además de en los ordenadores portátiles, resulta interesante gestionar las conexiones WiFi en aparatos más pequeños, como teléfonos móviles, consolas de videojuegos o PDAs, a los que se extiende este tipo de tecnología.

- CoovaSX ([35])
  Es una aplicación para dispositivos *Smartphone* compatible con MIDP-2.0. Admite el protocolo WISPr que facilita el roaming entre diferentes redes WiFi.
- HandyWi Es una buena opción para los modelos dotados del sistema operativo Symbian S60, como los Nokia, y funciona de manera similar a la anterior.

También existen dispositivos USB que de forma autónoma permiten detectar redes WiFi y mostrar sus características principales (SSID, canal, encriptación, estándar de transmisión y nivel de señal) a través de una pantalla LCD. Basta con conectar esta llave USB al dispositivo con el que se vaya a establecer la conexión WiFi.

- Allspot ([36])
  Se trata de un dispositivo USB con un adaptador WLAN y que además puede funcionar como localizador de puntos de acceso; dispone de una pantalla LCD en la que se muestran todos los datos de las redes wireless detectadas en un radio de 60 metros. Para este último modo de funcionamiento no es necesario estar conectado a un PC ya que dispone de una batería.
- TRENDnet TEW-453APB ([37])
  Es equivalente a la solución anterior pero funciona en un perímetro mayor (600 metros aproximadamente).

Las principales limitaciones de las soluciones descritas anteriormente respecto a **WiFi Locator** son:

1. El usuario puede conocer únicamente aquellas redes WiFi que tienen cobertura en su localización actual, pero no el resto.
2. El usuario conoce las redes disponibles sólo cuando ejecuta el software de forma explícita.
3. Necesidad de disponer de un equipo dotado con tecnología WiFi.

6.1.2. Bases de datos de APs WiFi

Estas aplicaciones utilizan la información relativa a puntos de acceso WiFi (también llamados *hotspots*) para que un usuario pueda conocer en qué puntos existe cobertura WiFi. Algunas de ellas se describen a continuación.

- WeFi ([38])
  Es una red social en la que la gente comparte puntos de acceso WiFi. El enfoque social de Wefi viene porque una vez que alguien ha accedido a una red abierta puede marcarla en un mapa para que otros puedan encontrarla después.
  Dispone de un software que permite gestionar las conexiones disponibles al alcance de la tarjeta inalámbrica clasificadas según su calidad. Existen versiones para Windows, Mac, Android, Symbian y Windows Mobile.
  La principal diferencia respecto a **WiFi Locator** es que la versión móvil de este software se conecta automáticamente al mejor *hotspot* disponible, pero no permite realizar búsquedas de otros puntos de acceso.

- FON ([29])
  Es una comunidad WiFi para la compartición de APs WiFi entre usuarios de dicha comunidad. Cuando un usario adquiere un router de FON (fonera) y lo conecta a su ADSL se crea un nuevo punto de acceso que puede ser accedido por otros usuarios. Todos los *hostspots* disponibles se pueden visualizar en su portal web.
  En comparación con la aplicación propuesta, FON se trata de una iniciativa empresarial y por tanto, si el usuario es miembro de la comunidad puede acceder gratuitamente a cualquier punto de acceso FON. Si por el contrario, no pertenece a la comunidad debe pagar una cuota para poder establecer la conexión.
  Otra diferencia es la vía de consulta de los *hotspots* disponibles; en la aplicación propuesta además de consultarlos en un mapa también es posible buscarlos desde un dispositivo Android.

- Otras basadas en Google Maps
  Dadas las facilidades que proporciona Google Maps para que un usuario pueda introducir puntos de interés en un mapa, existen varias iniciativas para registrar puntos de acceso WiFi de una región. Las principales desventajas respecto a **WiFi Locator** son:

  1. Suelen ser mapas con un ámbito regional (APs de Madrid por ejemplo) pero no existe una base de datos global.
  2. Almacenan la información general de los puntos de acceso pero no la cobertura de cada uno de ellos (al menos el mapa no permite visualizarla).
  3. No permiten importar APs de otros sistemas.

4. La consulta de los *hotspots* no es posible desde otros dispositivos, sólo a través de la herramienta basada en Google Maps.

Existen aplicaciones Android tales como **WiFi Finder** ([40]) o **Closest WiFi Hotspot Locator** ([39]), que ofrecen una funcionalidad similar a la solución que se propone en este PFC (**WiFi Locator**). Dichas aplicaciones permiten buscar los puntos de acceso WiFi más próximos a la localización actual del usuario y los más próximos a una localidad específica. Sin embargo, se han detectado los siguientes puntos de desventaja frente a **WiFi Locator**:

1. No están integradas con una herramienta web. Si lo están, son plataformas en fase beta y tienen algunas limitaciones: los usuarios no pueden añadir APs desde el mapa, no pueden especificar parámetros tales como potencia/velocidad, etc.
2. No avisan al usuario cada vez que entra en el área de cobertura de algún AP.
3. Manejan un número limitado de puntos de acceso y no permiten importar APs de otros sistemas (FON, etc).
4. No permiten realizar búsquedas de APs en base a la potencia o velocidad.
5. En el mapa donde se visualizan los APs no se proporciona información del radio de cada uno de ellos.
6. No permiten configurar el número de resultados a mostrar.
7. Existen acuerdos con partners, no son iniciativas libres.

En estas aplicaciones se establecen categorías para los APs (hotel, restaurante, etc). Este aspecto no se ha contemplado en **WiFi Locator** por considerarse de poca relevancia.

6.2. Conclusiones

La investigación de las tecnologías a emplear, el análisis y diseño de la aplicación, la codificación y el periodo de pruebas han sido las principales fases del presente proyecto. Como se ha podido comprobar a lo largo de la presente memoria, los objetivos propuestos en el capítulo de introducción han sido cumplidos satisfactoriamente. A continuación se revisa el estado de los dos grandes bloques o enunciados:

- Desarrollo de una interfaz web que permita gestionar puntos de acceso WiFi
  Se ha implementado una aplicación RIA (empleando tecnología Flex) que permite al usuario añadir APs en un punto determinado del mapa o en una dirección postal concreta. Asimismo el usuario (desde el navegador web) puede encontrar el AP más próximo a una dirección específica.

- Desarrollo de un módulo de búsqueda de redes WiFi para dispositivos Android.
  Se ha conseguido dar cobertura a distintos casos de búsqueda de redes WiFi: por proximidad a la localización actual, por proximidad a una dirección específica o por potencia/velocidad (en una distancia máxima).
  Para determinar la localización de un usuario se ha elegido la técnica de posicionamiento basada en GPS por ser la más adecuada, atendiendo a su coste, sencillez y precisión. No obstante se han investigado otros métodos disponibles en los dispositivos Android.

Después de haber analizado las características de Flex se puede decir que tiene un buen futuro por delante. La principal ventaja es que Flex dispone de gran cantidad de ejemplos, tutoriales y documentación disponible en la página oficial de Adobe, en los numerosos blogs de desarrolladores Flex y en alguna lista de distribución de correos como la de flexcoders.org.

Sin embargo, el entorno de programación *Flex Builder* no está preparado para desarrollar proyectos de gran envergadura. Por ello, la gente de Adobe se ha unido a la fundación Eclipse para crear un entorno de desarrollo para aplicaciones Flex basado en este IDE y cuyo nombre será *Zorn*.

Respecto al fututo de las RIA, cada vez más las grandes empresas están enfocando sus negocios entorno a este tipo de aplicaciones ya que al ser ejecutadas on-line ofrecen más ventajas que las aplicaciones tradicionales. Bien es cierto que la potencia de las aplicaciones tradicionales no es comparable a las aplicaciones web, pero este problema está siendo solventado poco a poco gracias a los plug-in en la parte de cliente y a la transferencia optimizada de datos entre cliente y servidor.

Por otro lado, se puede afirmar que Android será una de las plataformas móviles más relevantes del futuro porque:

- Es Open Source y gracias a ello cualquier persona puede contribuir con nuevas mejoras que ayuden a desarrollarla. Además los bugs detectados son corregidos a las pocas horas o días, no es necesario esperan una actualización que sale cada dos meses.

- Tiene el respaldo de las empresas de tecnología y telefonía celular más importantes del mundo, lo que garantiza la continuidad del proyecto.

- Permite que cualquier desarrollador publique sus aplicaciones, sin ninguna restricción ni evaluación del código fuente de la misma, al contrario de lo que ocurre en Apple, donde cada aplicación es examinada en detalle antes de ser puesta a disposición del público.

- Android tiene mucho más expectativas de crecimiento en los próximos años que otras plataformas, sobre todo cuando empiecen a aparecer nuevos celulares y dispositivos móviles que la utilicen.

6.3. Líneas futuras

Como continuación de este PFC, en la aplicación móvil pueden seguirse las siguientes líneas de investigación:

- Localización de dispositivos móviles en interiores
  En la aplicación Android desarrollada se ofrecen varios criterios de búsqueda de redes WiFi. Uno de ellos es la proximidad a la localización actual del usuario, calculada con la técnica de posicionamiento basada en GPS. Dicha técnica es válida en exteriores pero no sirve a la hora de establecer un seguimiento preciso en el interior de edificios. Las otras técnicas, por su parte, ofrecen un radio de error demasiado grande para establecer una posición.

- Obtención de APs de forma dinámica
  En el arranque de la aplicación Android desarrollada en este PFC se obtienen los APs de base de datos (compartida con la aplicación web) y se cachean. Se podría implementar un refresco dinámico de los APs para que cuando un usuario entre en el área de cobertura de alguno de ellos o realice una búsqueda explícita (en base a alguno de los criterios disponibles) obtenga la información más actualizada, no los datos cacheados.

- Visualización en el mapa del área de los APs
  Para que el usuario tenga una idea más exacta de la cobertura de los APs obtenidos (sea cual sea el criterio de búsqueda), se podría dibujar el área (circular) que depende del radio del AP y el nivel de zoom aplicado sobre el mapa. Esto solo aplica al componente Android, en la aplicación web sí se muestra el área de los APs.

- Instalación en un dispositivo Android
  Para probar con más exactitud las prestaciones de la aplicación diseñada, se debería instalar dicha aplicación en un terminal Android real.

Respecto a la aplicación web, algunas sugerencias son:

- Integración con otros sistemas
  Con el objetivo de hacer aún más flexible la aplicación web desarrollada en este PFC se podría añadir una funcionalidad que permitiese importar información de otros sistemas similares a FON.

- Exportación de APs
  Para que otras aplicaciones puedan acceder a los datos disponibles en esta herramienta se podría implementar una funcionalidad que permita exportar la información de los APs a fichero (con formato configurable).

- Configuración de redes *ad-hoc*
  La aplicación web desarrollada en este PFC permite añadir sobre el mapa puntos de acceso para que los usuarios se puedan conectar a distintas redes WiFi. Para aumentar la versatilidad de la herramienta se podría implementar una funcionalidad de soporte a redes *ad-hoc*. Una red de este tipo permite conectar equipos (con adaptador inalámbrico) sin necesidad de un punto de acceso. Todos los adaptadores inalámbricos de la red *ad-hoc* deben utilizar el mismo SSID y el mismo número de canal.

6.4. Presupuesto

El presupuesto de este proyecto, de duración 6 meses, consta de dos apartados: ejecución material y mano de obra.

6.4.1. Ejecución material

En este apartado se incluyen los gastos en herramientas empleadas, tanto hardware como software.

Concepto	**Precio (euros)**
Ordenador Windows XP	450
Licencia Office 2003	75
Impresora Láser	150
Material de oficina	150
TOTAL	**825**

Cuadro 6.1: Ejecución material

Las otras herramientas software empleadas no se contabilizan porque son de libre distribución.

6.4.2. Mano de obra

En este apartado se incluyen los gastos que suponen los recursos humanos que intervienen en la gestión y desarrollo del proyecto y en la elaboración de la memoria.

Concepto	Salario mensual bruto (euros)	Meses	Total (euros)
Ing Telecom. (gestión)	2000	4	8000
Ing. téc. Telecom. (desarrollo)	1750	3	5250
Mecanógrafo	1000	1	1000
TOTAL			**14250**

Cuadro 6.2: Mano de obra

Este precio se pondera por 1.5 para dar cobertura a los gastos indirectos de los recursos citados anteriormente, alcanzando los 21375 euros.

Sumando los conceptos anteriores se llega al importe total de proyecto.

El importe total asciende a VEINTE DOS MIL DOSCIENTOS EUROS

Fdo: Patricia Bravo García

Madrid, Enero de 2010

Apéndice A

Código Flex

A.1. Fichero pfc.mxml

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
                xmlns:controller="es.uc3m.pfc.controller.*"
                xmlns:view="es.uc3m.pfc.view.*"
                backgroundColor="#BBBBBB"
                layout = "vertical"
                paddingRight="50"
                paddingLeft="50">
  <mx:Style source="pfc.css"/>

  <controller:PfcController/>

</mx:Application>
```

A.2. Clase PfcService.as

```
package es.uc3m.pfc.business
{
  import es.uc3m.pfc.model.Model;
  import es.uc3m.pfc.model.AP;
  import mx.collections.ArrayCollection;
  import mx.rpc.AsyncToken;
  import mx.rpc.IResponder;
  import mx.rpc.remoting.mxml.RemoteObject;

  public class PfcService{
    protected static var pfcService:PfcService;
    private var remoteObject:RemoteObject;
    private var model:Model;

    public static function getInstance():PfcService {
      if(pfcService == null) {
        pfcService = new PfcService();
      }
```

```
19        return pfcService;
20      }
21
22      public function PfcService(){
23        model = Model.getInstance();
24        remoteObject = new RemoteObject("PfcService");
25        remoteObject.showBusyCursor = true;
26      }
27
28      public function authenticate(responder:IResponder,
29                                   user:String, password:String):void {
30        var token:AsyncToken = remoteObject.authenticate(user,password);
31        token.addResponder(responder);
32      }
33
34      public function register(responder:IResponder, user:String, password:String):void {
35        var token:AsyncToken = remoteObject.register(user,password);
36        token.addResponder(responder);
37      }
38
39      public function getAllAPs(responder:IResponder):void {
40        var token:AsyncToken = remoteObject.getAllAPs();
41        token.addResponder(responder);
42      }
43
44      public function saveAP(responder:IResponder, ap:AP):void {
45        var token:AsyncToken = remoteObject.saveAP(ap);
46        token.addResponder(responder);
47      }
48
49      public function removeAP(responder:IResponder, apToRemove:AP):void {
50        var token:AsyncToken = remoteObject.removeAP(apToRemove);
51        token.addResponder(responder);
52      }
53
54      public function removeAllAPs(responder:IResponder):void {
55        var token:AsyncToken = remoteObject.removeAllAPs();
56        token.addResponder(responder);
57      }
58
59      public function importAPsFon(responder:IResponder):void {
60        var token:AsyncToken = remoteObject.importAPsFon();
61        token.addResponder(responder);
62      }
63
64      public function searchAPs(responder:IResponder, latitudeToSearch:Number,
65                                longitudeToSearch:Number):void {
66        var token:AsyncToken = remoteObject.searchAPs(
67                               latitudeToSearch, longitudeToSearch);
68        token.addResponder(responder);
69      }
70
71      public function dragAP(responder:IResponder, apToDrag:AP):void {
72        var token:AsyncToken = remoteObject.dragAP(apToDrag);
73        token.addResponder(responder);
74      }
```

```
75    }
76  }
```

A.3. Clase RegisterUserCommand.as

```
1  package es.uc3m.pfc.business
2  {
3    import es.uc3m.pfc.events.LogoutEvent;
4    import es.uc3m.pfc.model.Model;
5
6    import flash.events.Event;
7
8    import mx.controls.Alert;
9    import mx.managers.PopUpManager;
10   import mx.rpc.IResponder;
11   import mx.rpc.events.FaultEvent;
12   import mx.rpc.events.ResultEvent;
13
14   import org.osflash.thunderbolt.Logger;
15
16   public class RegisterUserCommand implements Command, IResponder{
17     private var model:Model = Model.getInstance();
18     private var pfcService:PfcService = PfcService.getInstance();
19
20     public function execute(event:Event):void {
21       pfcService.register(this, model.login, model.password);
22     }
23
24     public function result(event:Object):void {
25       var result:int = int(ResultEvent(event).result);
26       Alert.show("Usuario registrado correctamente", "Aviso");
27       // cierra el popup
28       PopUpManager.removePopUp(model.registerUserPopup);
29       // ir a la pantalla de login
30       new LogoutCommand().execute(new LogoutEvent());
31     }
32
33     public function fault(event:Object):void {
34       Alert.show("Error en llamada a servidor","Aviso");
35       Logger.info("Error en llamada a servidor: "+FaultEvent(event).fault.message,
36       FaultEvent(event).fault);
37     }
38   }
39 }
```

A.4. Clase RegisterUserEvent.as

```
1  package es.uc3m.pfc.events
2  {
3    import flash.events.Event;
```

```
4
5     public class RegisterUserEvent extends Event{
6       public static const EVENT_ID:String = "registerUser";
7       public function RegisterUserEvent() {
8         super(EVENT_ID, true);
9       }
10    }
11  }
```

A.5. Clase AP.as

```
1   package es.uc3m.pfc.model {
2     import flash.utils.IDataInput;
3     import flash.utils.IDataOutput;
4     import flash.utils.IExternalizable;
5     [Bindable]
6     public class APBase implements IExternalizable {
7       private var _cyphered:Boolean;
8       private var _fon:Boolean;
9       private var _lat:Number;
10      private var _lng:Number;
11      private var _name:String;
12      private var _radius:Number;
13      private var _txPower:Number;
14      private var _velocity:Number;
15
16      public function readExternal(input:IDataInput):void {
17        _cyphered = input.readObject() as Boolean;
18        _fon = input.readObject() as Boolean;
19        _lat = function(o:*):Number {
20          return (o is Number ? o as Number : Number.NaN)
21        } (input.readObject());
22        _lng = function(o:*):Number {
23          return (o is Number ? o as Number : Number.NaN)
24        } (input.readObject());
25        _name = input.readObject() as String;
26        _radius = function(o:*):Number {
27          return (o is Number ? o as Number : Number.NaN)
28        } (input.readObject());
29        _txPower = function(o:*):Number {
30          return (o is Number ? o as Number : Number.NaN)
31        } (input.readObject());
32        _velocity = function(o:*):Number {
33          return (o is Number ? o as Number : Number.NaN)
34        } (input.readObject());
35      }
36
37      public function writeExternal(output:IDataOutput):void {
38        output.writeObject(_cyphered);
39        output.writeObject(_fon);
40        output.writeObject(_lat);
41        output.writeObject(_lng);
42        output.writeObject(_name);
```

```
43          output.writeObject(_radius);
44          output.writeObject(_txPower);
45          output.writeObject(_velocity);
46        }
47      }
48  }
```

A.6. Clase Util.as

```
1   package es.uc3m.pfc.util
2   {
3     import es.uc3m.pfc.events.DragAPEvent;
4     import es.uc3m.pfc.events.RemoveAPEvent;
5     import es.uc3m.pfc.events.RemoveAllAPsEvent;
6     import es.uc3m.pfc.business.RemoveAPCommand;
7     import es.uc3m.pfc.business.RemoveAllAPsCommand;
8     import es.uc3m.pfc.business.DragAPCommand;
9     import es.uc3m.pfc.model.Model;
10    import es.uc3m.pfc.model.AP;
11    import es.uc3m.pfc.view.ModifyAccessPointPopup;
12    import flash.display.DisplayObject;
13    import flash.geom.Point;
14    import flash.utils.Dictionary;
15    import mx.collections.ArrayCollection;
16    import mx.collections.IViewCursor;
17    import mx.controls.Alert;
18    import mx.core.Application;
19    import mx.core.IFlexDisplayObject;
20    import mx.managers.PopUpManager;
21    import com.google.maps.overlays.MarkerOptions;
22    import com.google.maps.overlays.Polygon;
23    import com.google.maps.overlays.PolygonOptions;
24    import com.google.maps.LatLng;
25    import com.google.maps.styles.StrokeStyle;
26    import com.google.maps.styles.FillStyle;
27    import com.google.maps.MapMouseEvent;
28
29    public class Util {
30
31      [Bindable]
32      private static var model:Model = Model.getInstance();
33
34      [Embed(source="/assets/images/antena.png")]
35      private static var imgCls:Class
36
37      public static var arrayCollection1:ArrayCollection = new ArrayCollection();
38      public static var arrayCollection2:ArrayCollection = new ArrayCollection();
39      public static var arrayCollection3:ArrayCollection = new ArrayCollection();
40      public static var arrayCollection4:ArrayCollection = new ArrayCollection();
41      public static var arrayCollection5:ArrayCollection = new ArrayCollection();
42      public static var arrayCollection6:ArrayCollection = new ArrayCollection();
43      public static var arrayCollection7:ArrayCollection = new ArrayCollection();
44
```

```
45      public static function getMarkerOptions():MarkerOptions{
46        var markerOptions:MarkerOptions = new MarkerOptions();
47        markerOptions.draggable=true;
48
49        markerOptions.icon = new imgCls();
50        markerOptions.iconAlignment =
51        MarkerOptions.ALIGN_BOTTOM | MarkerOptions.ALIGN_HORIZONTAL_CENTER;
52        markerOptions.distanceScaling = true;
53        markerOptions.gravity = 50;
54        markerOptions.iconOffset = new Point(2, 2);
55        markerOptions.strokeStyle=new StrokeStyle({color: 0x987654});
56        markerOptions.fillStyle=new FillStyle({color: 0x223344,alpha: 0.8});
57        markerOptions.radius=12;
58        markerOptions.hasShadow=true;
59
60        return markerOptions;
61      }
62
63      public static function removeAllAPMarkers():void{
64        model.map.clearOverlays();
65        model.apMarkerAreaDictionary=new Dictionary();
66
67        // elimina de base de datos todos los APs
68        new RemoveAllAPsCommand().execute(new RemoveAllAPsEvent());
69      }
70
71      public static function removeAPMarker():void{
72
73        // elimina area del mapa
74        model.markerManager.removeMarker(model.selectedAPMarker);
75
76        var ap:AP = new AP();
77        ap.name=model.selectedAPMarker.name;
78        ap.lat=model.selectedAPMarker.lat;
79        ap.lng=model.selectedAPMarker.lng;
80        ap.radius=model.selectedAPMarker.radius;
81        ap.txPower=model.selectedAPMarker.txPower;
82        ap.cyphered=model.selectedAPMarker.cyphered;
83        ap.fon=model.selectedAPMarker.fon;
84        model.ap = ap;
85        model.apMarkerAreaDictionary[model.selectedAPMarker.name]=null;
86
87        // elimina de base de datos el AP seleccionado
88        new RemoveAPCommand().execute(new RemoveAPEvent());
89
90      }
91
92      public static function addAPMarker(name:String, radius:Number,
93                                         lat:Number, lng:Number,
94                                         txPower:Number, velocity:Number,
95                                         cyphered:Boolean):APMarker{
96
97        // crea marker, fon =false
98        var newAPMarker:APMarker = new APMarker(getMarkerOptions(), name, radius, lat, lng,
99                                                txPower, velocity, cyphered, false);
100
```

```
101         // solo se puede hacer drag start/drag end/double click si el role es 'write'
102
103         if(model.role == 'write'){
104           // drag del marker
105           newAPMarker.addEventListener(MapMouseEvent.DRAG_START,
106             function(event:Event):void {
107               model.selectedAPMarker = APMarker(MapMouseEvent(event).currentTarget);
108               var area:Polygon = model.apMarkerAreaDictionary[model.selectedAPMarker.name];
109               // elimino area del mapa
110               model.map.removeOverlay(area);
111               // para distinguir los markers que han sido eliminados
112               model.apMarkerAreaDictionary[model.selectedAPMarker.name]=null;
113             });
114
115           newAPMarker.addEventListener(MapMouseEvent.DRAG_END,
116             function(event:Event):void {
117               model.selectedAPMarker = APMarker(MapMouseEvent(event).currentTarget);
118               // anade area al mapa
119               var area:Polygon = getArea(MapMouseEvent(event).latLng,
120                                          model.selectedAPMarker.radius);
121               model.map.addOverlay(area);
122               // marker-area en el dictionary
123               model.apMarkerAreaDictionary[model.selectedAPMarker.name]=area;
124               // solo necesito estos datos para hacer drag
125               model.ap = new AP();
126               model.ap.name=model.selectedAPMarker.name;
127               model.ap.lat=MapMouseEvent(event).latLng.lat();
128               model.ap.lng=MapMouseEvent(event).latLng.lng();
129               new DragAPCommand().execute(new DragAPEvent());
130             });
131
132           // doble click
133           newAPMarker.addEventListener(MapMouseEvent.DOUBLE_CLICK,
134             function(event:Event):void {
135               model.selectedAPMarker=APMarker(MapMouseEvent(event).currentTarget);
136               // popup para EDITAR, BORRAR
137               var displayObject:IFlexDisplayObject =
138               PopUpManager.createPopUp(DisplayObject(Application.application),
139                                          ModifyAccessPointPopup,true);
140               PopUpManager.centerPopUp(displayObject);
141             });
142         }
143
144         var coordinatesUtil:CoordinatesUtil = new CoordinatesUtil();
145         // simple click
146         newAPMarker.addEventListener(MapMouseEvent.CLICK, function(event:Event):void {
147           coordinatesUtil.coordinatesToStreetName(newAPMarker);
148         });
149
150         // roll over
151         newAPMarker.addEventListener(MapMouseEvent.ROLL_OVER,
152           function(event:Event):void {
153             coordinatesUtil.coordinatesToStreetName(newAPMarker);
154           });
155
156         // roll out
```

```
157         newAPMarker.addEventListener(MapMouseEvent.ROLL_OUT,
158           function(event:Event):void {
159             newAPMarker.closeInfoWindow();
160           });
161
162         return newAPMarker;
163       }
164
165       public static function getArea(center:LatLng, radius:Number):Polygon {
166
167         var colors:Array = [0x808080, 0x808080, 0x808080];
168         var color:Object = colors[Math.round(Math.random()*colors.length)];
169         var d2r:Number = Math.PI/180;
170         var r2d:Number = 180/Math.PI;
171         // de metros a statute miles --> dividir por 1000 para pasar a Km
172         // dividir por 1.60934 para pasar a statute miles
173         // Convert statute miles = 1.60934Km into degrees latitude
174         var circleLat:Number = (radius/1000/1.60934) * 0.014483 ;
175         var circleLng:Number = circleLat/Math.cos(center.lat()*d2r);
176
177         var circleLatLngs:Array = new Array();
178         for (var i:Number = 0; i < 33; i++) {
179           var theta:Number = Math.PI * (i/16);
180           var vertexLat:Number = center..lat() + (circleLat * Math.sin(theta));
181           var vertexLng:Number = center..lng() + (circleLng * Math.cos(theta));
182           var latLng:LatLng = new LatLng(vertexLat, vertexLng);
183           circleLatLngs.push(latLng);
184         }
185
186         var polygon:Polygon = new Polygon(circleLatLngs, new PolygonOptions (
187         {
188           strokeStyle: new StrokeStyle({alpha:0.7,color:0x808080,thickness:3,
189           pixelHinting: true}), fillStyle: new FillStyle({alpha:0.2, color:0x808080})
190          }));
191
192         return polygon;
193       }
194
195       public static function init():void{
196
197         model.apMarkerAreaDictionary = new Dictionary();
198
199         arrayCollection1 = new ArrayCollection();
200         arrayCollection2 = new ArrayCollection();
201         arrayCollection3 = new ArrayCollection();
202         arrayCollection4 = new ArrayCollection();
203         arrayCollection5 = new ArrayCollection();
204         arrayCollection6 = new ArrayCollection();
205         arrayCollection7 = new ArrayCollection();
206
207         model.apNamesList = new ArrayCollection();
208
209         model.markerManager.clearMarkers();
210         model.map.setCenter(new LatLng(0,0));
211         model.map.setZoom(1);
212
```

```
213       }
214
215       public static function load():void{
216
217         var index:int=0;
218
219         for each(var ap:AP in model.apList){
220           // nombres de los APs (no repetidos)
221           model.apNamesList.addItem(ap.name);
222
223           var apMarker:APMarker = addAPMarker(ap.name, ap.radius, ap.lat, ap.lng,
224                                               ap.txPower, ap.velocity, ap.cyphered);
225           model.apMarkerAreaDictionary[apMarker.name]=Util.getArea(
226                                                   new LatLng(apMarker.lat,
227                                                              apMarker.lng),
228                                                   apMarker.radius);
229           if(index<100){
230             arrayCollection1.addItem(apMarker);
231           }
232           else if(index<200){
233             arrayCollection2.addItem(apMarker);
234           }
235           else if(index<400){
236             arrayCollection3.addItem(apMarker);
237           }
238           else if(index<800){
239             arrayCollection4.addItem(apMarker);
240           }
241           else if(index<1600){
242             arrayCollection5.addItem(apMarker);
243           }
244           else if(index<3200){
245             arrayCollection6.addItem(apMarker);
246           }
247           else{
248             arrayCollection7.addItem(apMarker);
249           }
250           index=index+1;
251         }
252         model.markerManager.addMarkers(arrayCollection1,2);
253         model.markerManager.addMarkers(arrayCollection2,3);
254         model.markerManager.addMarkers(arrayCollection3,4);
255         model.markerManager.addMarkers(arrayCollection4,5);
256         model.markerManager.addMarkers(arrayCollection5,6);
257         model.markerManager.addMarkers(arrayCollection6,7);
258         model.markerManager.addMarkers(arrayCollection7,8);
259       }
260     }
261 }
```

A.7. Fichero HeaderPanel.mxml

```
1 <?xml version="1.0" encoding="utf-8"?>
```

```
2  <mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="100%" verticalGap="1">
3    <mx:Script>
4    <![CDATA[
5      import es.uc3m.pfc.events.LogoutEvent;
6      import es.uc3m.pfc.model.Model;
7      import mx.controls.Alert;
8      import mx.events.CloseEvent;
9      [Bindable]
10     private var model:Model = Model.getInstance();
11     private function logout():void{
12       Alert.yesLabel = "Si"
13       Alert.noLabel = "No"
14       Alert.show("Desea realmente salir de la aplicacion",
15       "Confirmacion", 3, this, alertClickHandler);
16     }
17     private function alertClickHandler(event:CloseEvent):void {
18       if (event.detail==Alert.YES){
19         dispatchEvent(new LogoutEvent());
20       }
21     }
22   ]]>
23   </mx:Script>
24
25   <!-- linea verde -->
26   <mx:HBox backgroundColor="0x007087" width="100%"
27   horizontalAlign="right" height="30">
28     <mx:Text text="Usuario: " fontSize="12" color="0xFFFFFF"
29     fontWeight="bold" textAlign="right" visible="{model.selectedIndex!=0}"/>
30     <mx:Text text="{model.login}"  fontSize="12" color="0xFFFFFF"
31     textAlign="right" visible="{model.selectedIndex!=0}"/>
32     <mx:LinkButton label="Salir" fontSize="12" color="0xFFFFFF"
33     fontWeight="bold" click="logout()" visible="{model.selectedIndex!=0}"/>
34   </mx:HBox>
35
36   <!-- linea azul -->
37   <mx:HBox backgroundColor="0x000080" width="100%"
38     paddingLeft="15" paddingTop="5" paddingBottom="5">
39     <mx:Image source="@Embed('/assets/images/header.png')" width="10%"/>
40     <mx:VBox width="50%">
41       <mx:Text width="100%" textAlign="center" color="0xffffff"
42       fontSize="23" fontWeight="bold" text="Universidad Carlos III de Madrid"/>
43       <mx:Text width="100%" textAlign="center" color="0xffffff"
44       fontSize="20" fontWeight="bold" text="Dpto. Ing. Telematica"/>
45     </mx:VBox>
46     <!-- relleno -->
47     <mx:VBox width="40%"/>
48   </mx:HBox>
49
50 </mx:VBox>
```

Apéndice B

Bugs del emulador de Android

Durante el desarrollo de la aplicación Android (versión 1.6 r1) se detectaron dos bugs. El primero de ellos está relacionado con el formato de las coordenadas enviadas al emulador. Por algún motivo (probablemente relacionado con el carácter usado para separa los decimales, la coma en español y punto en inglés) el emulador sólo recibe correctamente las actualizaciones si la configuración regional está establecida a idioma inglés.

El otro bug detectado ocurre bajo el siguiente escenario:

- Lanzar una aplicación con el emulador incluido en el SDK de Android.
- Ejecutar por línea de comandos las instrucciones necesarias para enviar actualizaciones de localización al emulador o utilizar la herramienta DDMS (descrita en el Capítulo 3).
- Volver a paso 2.

Por algún motivo, las localizaciones enviadas después de la primera no llegan al emulador. Se descubrió que la causa del problema era el uso de un zona horaria errónea en el controlador GPS del driver. La solución para evitar dicho problema es configurar la zona horaria adecuadamente:

- En la pantalla principal del emulador seleccionar la opción *Menú* $->$ *Settings* $->$ *Date Time* (ver Figura B.1).
- Desmarcar la opción *Automatic* (ver Figura B.2).
- Pulsar en la opción *Select Time Zone* y elegir la zona horaria GMT+1 (ver Figura B.3).

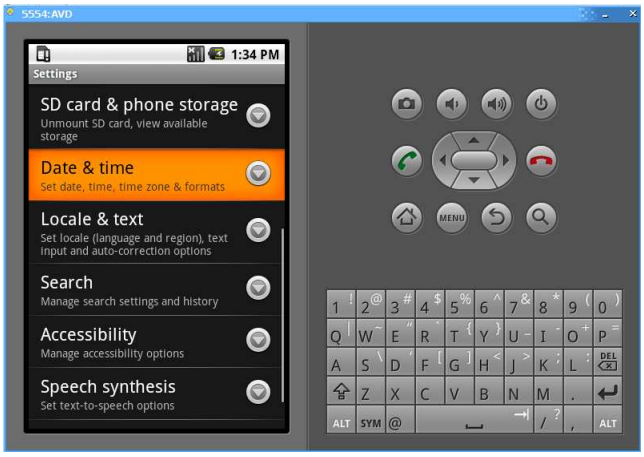

Figura B.1: Fecha y hora del dispositivo Android

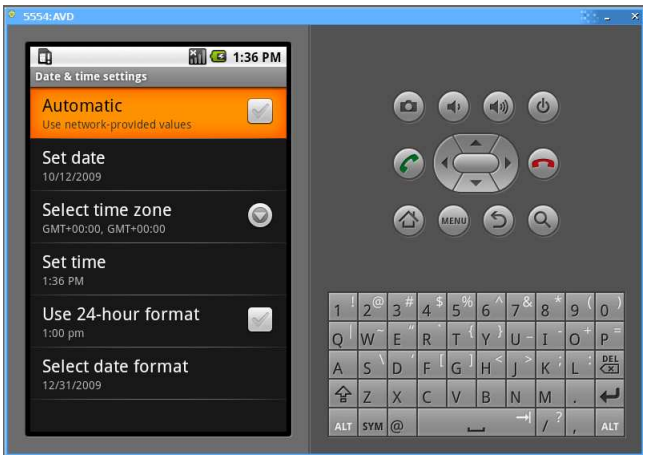

Figura B.2: Desmarcar la opción "Automatic"

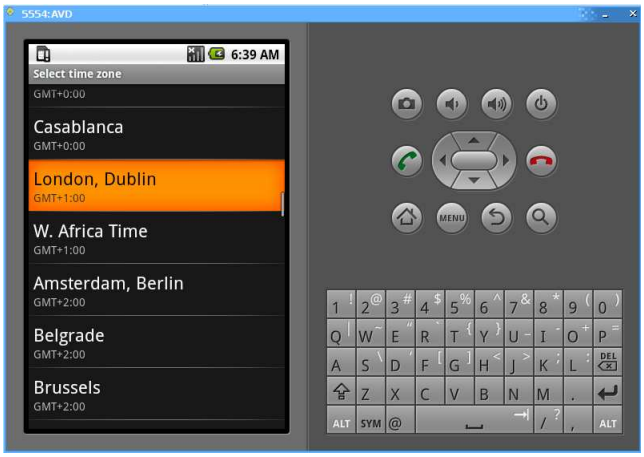

Figura B.3: Seleccionar zona GMT+1

Bibliografía

[1] www.wikipedia.es

[2] www.as3dp.com

[3] www.adobe.com/devnet/flex/articles/blueprint.html

[4] www.adobe.com/devnet/actionscript/articles/actionscript3_overview.html

[5] www.adobe.com/devnet/flex/quickstart/coding_with_mxml_and_actionscript

[6] learn.adobe.com/wiki/display/Flex/Getting+Started

[7] livedocs.adobe.com/flex/3

[8] www.aboutflex.net

[9] www.afcomponents.com/blog

[10] www.adobe.com/devnet/flash/articles/mv_controller.html

[11] www.onjava.com/pub/a/onjava/2004/12/01/flexjava.html

[12] www.theopensourcery.com/xmlria.htm

[13] www.devarticles.com/c/a/Flash/Introduction-to-Flex

[14] java.sun.com/blueprints/patterns/MVC-detailed.html

[15] www.graniteds.org

[16] code.google.com/apis/maps

[17] code.google.com/apis/gdata

[18] code.google.com/webtoolkit

[19] www.android.com

[20] www.openhandsetalliance.com

[21] code.google.com/Android

[22] developer.android.com/guide/tutorials/views/hello-mapview.html

[23] www.droiddraw.org

[24] www.anddev.org

[25] www.Android-spa.com

[26] groups.google.com/group/Android-developers/files

[27] www.balmat.es/queesgps.asp

[28] www.wi-fi.org

[29] www.fon.com/es

[30] www.maestrosdelweb.com

[31] tomcat.apache.org

[32] www.mysql.com

[33] java.sun.com/javase/technologies/database

[34] www.xirrus.com

[35] www.coova.com/CoovaSX

[36] www.laflecha.net/gadgets/allspot_-busqueda-de-redes-wi-fi

[37] www.trendnet.com

[38] www.wefi.com

[39] es.androlib.com/android.application.com-ochosi-wifinderlite-qzF.aspx

[40] es.androlib.com/android.application.com-jiwire-android_apps-finder-zqqj.aspx